

La matematica della crittografia moderna

Parte I: Fondamenti Teorici della Teoria dei Numeri

La crittografia moderna si fonda su pilastri matematici profondi che affondano le loro radici nella teoria dei numeri, nell'algebra astratta e nella teoria della complessità computazionale. Tratteremo la maggior parte degli aspetti matematici che sottendono i moderni sistemi crittografici, partendo dai teoremi fondamentali fino ad arrivare alle più recenti scoperte e alle loro applicazioni pratiche negli algoritmi di cifratura utilizzati quotidianamente.

La crittografia a chiave pubblica, che rappresenta il cuore pulsante della sicurezza informatica contemporanea, si basa su problemi matematici che sono computazionalmente difficili da risolvere in una direzione, ma relativamente semplici nell'altra. Questa asimmetria computazionale è ciò che rende possibile la comunicazione sicura su canali non protetti, la firma digitale dei documenti e l'autenticazione delle identità nel mondo digitale.

1. Teoria dei Numeri: I Fondamenti Classici

1.1 Il Piccolo Teorema di Fermat

Il Piccolo Teorema di Fermat rappresenta uno dei risultati più eleganti e fondamentali della teoria dei numeri elementare. Pierre de Fermat lo enunciò nel 1640, anche se la prima dimostrazione completa fu fornita da Leonhard Euler circa un secolo dopo.

Enunciato: Se p è un numero primo e a è un intero non divisibile per p , allora:

$$a^{(p-1)} \equiv 1 \pmod{p}$$

Una formulazione equivalente, ma più generale, afferma che per ogni intero a e per ogni primo p :

$$a^p \equiv a \pmod{p}$$

Questa seconda forma è valida anche quando a è divisibile per p , rendendo il teorema applicabile universalmente.

Dimostrazione completa del Piccolo Teorema di Fermat:

Consideriamo i primi $p-1$ multipli di a :

$$a, 2a, 3a, 4a, \dots, (p-1)a$$

Dobbiamo dimostrare che questi $p-1$ numeri, quando ridotti modulo p , formano una permutazione dei numeri $1, 2, 3, \dots, p-1$.

Passo 1: Nessuno di questi multipli è congruo a 0 modulo p

Se $ka \equiv 0 \pmod{p}$ per qualche k con $1 \leq k \leq p-1$, allora p divide ka . Poiché p è primo e non divide a per ipotesi, per il Lemma di Euclide p deve dividere k . Ma questo è impossibile perché $1 \leq k \leq p-1$, quindi k non può essere un multiplo di p .

Passo 2: Tutti i multipli sono distinti modulo p

Supponiamo che per due indici distinti i e j con $1 \leq i < j \leq p-1$ si abbia:

$$ia \equiv ja \pmod{p}$$

Questo implica che p divide $(j-i)a$. Poiché p è primo e non divide a , deve dividere $(j-i)$. Ma questo è impossibile perché $0 < j-i < p-1$, quindi $j-i$ non può essere un multiplo di p .

Passo 3: Conclusione

Abbiamo dimostrato che i numeri $a, 2a, 3a, \dots, (p-1)a$ formano, modulo p , una permutazione di $1, 2, 3, \dots, p-1$. Quindi il loro prodotto è congruo al prodotto dei numeri da 1 a $p-1$:

$$a \cdot 2a \cdot 3a \cdot \dots \cdot (p-1)a \equiv 1 \cdot 2 \cdot 3 \cdot \dots \cdot (p-1) \pmod{p}$$

Possiamo riscrivere il lato sinistro come:

$$a^{(p-1)} \cdot (1 \cdot 2 \cdot 3 \cdot \dots \cdot (p-1)) \equiv (1 \cdot 2 \cdot 3 \cdot \dots \cdot (p-1)) \pmod{p}$$

Sia $(p-1)! = 1 \cdot 2 \cdot 3 \cdot \dots \cdot (p-1)$. Poiché p è primo, nessuno dei fattori di $(p-1)!$ è divisibile per p , quindi $(p-1)!$ non è divisibile per p . Questo significa che $(p-1)!$ ha un inverso moltiplicativo modulo p . Moltiplicando entrambi i lati per questo inverso, otteniamo:

$$a^{(p-1)} \equiv 1 \pmod{p}$$

Questo completa la dimostrazione del Piccolo Teorema di Fermat.

Applicazioni crittografiche immediate:

Il Piccolo Teorema di Fermat è fondamentale per la costruzione di molti sistemi crittografici. In particolare, esso garantisce che, lavorando modulo un numero primo p , ogni elemento non nullo ha un ordine che divide $p-1$. Questo fatto è alla base della definizione di generatori nei gruppi ciclici finiti, concetto cruciale per i protocolli di scambio delle chiavi come Diffie-Hellman.

Inoltre, il teorema fornisce un metodo efficiente per calcolare gli inversi moltiplicativi modulo p . Se vogliamo trovare l'inverso di a modulo p , sappiamo che:

$$a \cdot a^{(p-2)} \equiv a^{(p-1)} \equiv 1 \pmod{p}$$

Quindi $a^{(p-2)}$ è l'inverso moltiplicativo di a modulo p . Questo può essere calcolato efficientemente usando l'algoritmo di esponenziazione veloce, con complessità $O(\log p)$ moltiplicazioni modulari.

1.2 Il Teorema di Eulero

Il Teorema di Eulero rappresenta una generalizzazione del Piccolo Teorema di Fermat a moduli composti, e introduce la funzione φ di Eulero (o funzione toziente), che è di importanza cruciale nella crittografia moderna, in particolare nell'algoritmo RSA.

La Funzione φ di Eulero:

Per un intero positivo n , la funzione $\varphi(n)$ conta il numero di interi positivi minori o uguali a n che sono coprimi con n (cioè che hanno massimo comun divisore con n uguale a 1).

Formalmente:

$$\varphi(n) = |\{k \in \mathbb{N} : 1 \leq k \leq n, \gcd(k,n) = 1\}|$$

Proprietà fondamentali della funzione φ :

1. Se p è un numero primo, allora $\varphi(p) = p - 1$, perché tutti i numeri da 1 a $p-1$ sono coprimi con p .
2. Se p è primo e $k \geq 1$, allora $\varphi(p^k) = p^k - p^{k-1} = p^{k-1}(p-1)$. Questo perché i soli numeri non coprimi con p^k sono i multipli di p , e ce ne sono esattamente p^{k-1} .
3. La funzione φ è moltiplicativa: se $\gcd(m,n) = 1$, allora $\varphi(mn) = \varphi(m)\varphi(n)$.

Dimostrazione della moltiplicatività di φ :

Questa proprietà è fondamentale e merita una dimostrazione completa. Supponiamo che $\gcd(m,n) = 1$. Dobbiamo dimostrare che $\varphi(mn) = \varphi(m)\varphi(n)$.

Consideriamo il sistema di congruenze:

$$x \equiv a \pmod{m} \quad x \equiv b \pmod{n}$$

dove $\gcd(a,m) = 1$ e $\gcd(b,n) = 1$.

Per il Teorema Cinese del Resto (che dimostreremo più avanti), questo sistema ha una soluzione unica modulo mn . Inoltre, questa soluzione x soddisfa $\gcd(x,mn) = 1$. Infatti, se un primo p dividesse sia x che mn , allora p divide m oppure p divide n (o entrambi). Ma se p divide m , allora $x \equiv a \pmod{m}$ implica che p divide a , contraddicendo $\gcd(a,m) = 1$. Analogamente se p divide n .

Viceversa, se $\gcd(x,mn) = 1$, allora $\gcd(x,m) = 1$ e $\gcd(x,n) = 1$. Ponendo $a \equiv x \pmod{m}$ e $b \equiv x \pmod{n}$, abbiamo che x è l'unica soluzione del sistema sopra con $\gcd(a,m) = 1$ e $\gcd(b,n) = 1$.

Quindi c'è una corrispondenza biunivoca tra:

- Le coppie (a,b) con $1 \leq a \leq m$, $1 \leq b \leq n$, $\gcd(a,m) = 1$, $\gcd(b,n) = 1$
- Gli interi x con $1 \leq x \leq mn$ e $\gcd(x,mn) = 1$

Il numero di tali coppie è $\varphi(m) \cdot \varphi(n)$, mentre il numero di tali x è $\varphi(mn)$. Quindi $\varphi(mn) = \varphi(m)\varphi(n)$.

Calcolo di φ per numeri composti:

Se n ha la fattorizzazione in primi:

$$n = p_1^{a_1} \cdot p_2^{a_2} \cdot \dots \cdot p_k^{a_k}$$

allora, per la moltiplicatività di φ :

$$\begin{aligned} \varphi(n) &= \varphi(p_1^{a_1}) \cdot \varphi(p_2^{a_2}) \cdot \dots \cdot \varphi(p_k^{a_k}) = p_1^{a_1-1}(p_1-1) \cdot p_2^{a_2-1}(p_2-1) \cdot \dots \cdot \\ p_k^{a_k-1}(p_k-1) &= n \cdot (1 - 1/p_1) \cdot (1 - 1/p_2) \cdot \dots \cdot (1 - 1/p_k) \end{aligned}$$

Questa formula è estremamente importante nella crittografia RSA, dove la conoscenza della fattorizzazione di n permette il calcolo efficiente di $\varphi(n)$.

Enunciato del Teorema di Eulero:

Se n è un intero positivo e a è un intero coprimo con n (cioè $\gcd(a,n) = 1$), allora:

$$a^{\varphi(n)} \equiv 1 \pmod{n}$$

Dimostrazione del Teorema di Eulero:

Sia $S = \{x_1, x_2, \dots, x_{\varphi(n)}\}$ l'insieme degli interi tra 1 e n che sono coprimi con n . Per definizione, $|S| = \varphi(n)$.

Passo 1: Il prodotto $ax \pmod{n}$ permuta S

Per ogni $x_i \in S$, consideriamo $ax_i \pmod{n}$. Dobbiamo dimostrare che: a) $\gcd(ax_i, n) = 1$ b) Se $i \neq j$, allora $ax_i \not\equiv ax_j \pmod{n}$

Per il punto (a): Se un primo p dividesse sia ax_i che n , allora p divide a oppure p divide x_i . Ma $\gcd(a,n) = 1$, quindi p non divide a . E $\gcd(x_i,n) = 1$, quindi p non divide x_i . Contraddizione.

Per il punto (b): Se $ax_i \equiv ax_j \pmod{n}$, allora n divide $a(x_i - x_j)$. Poiché $\gcd(a,n) = 1$, deve essere n che divide $(x_i - x_j)$. Ma $0 < |x_i - x_j| < n$, quindi questo è impossibile a meno che $x_i = x_j$.

Passo 2: Conclusione

I numeri $ax_1, ax_2, \dots, ax_{\varphi(n)}$, ridotti modulo n , formano una permutazione di $x_1, x_2, \dots, x_{\varphi(n)}$. Quindi:

$$(ax_1) \cdot (ax_2) \cdot \dots \cdot (ax_{\varphi(n)}) \equiv x_1 \cdot x_2 \cdot \dots \cdot x_{\varphi(n)} \pmod{n}$$

$$a^{\varphi(n)} \cdot (x_1 \cdot x_2 \cdot \dots \cdot x_{\varphi(n)}) \equiv x_1 \cdot x_2 \cdot \dots \cdot x_{\varphi(n)} \pmod{n}$$

Sia $P = x_1 \cdot x_2 \cdot \dots \cdot x_{\varphi(n)}$. Poiché ogni x_i è coprimo con n , anche il loro prodotto P è coprimo con n , quindi P ha un inverso moltiplicativo modulo n . Moltiplicando entrambi i lati per P^{-1} , otteniamo:

$$a^{\varphi(n)} \equiv 1 \pmod{n}$$

Significato crittografico profondo:

Il Teorema di Eulero è il cuore matematico dell'algoritmo RSA. In RSA, si sceglie $n = pq$ dove p e q sono primi distinti e grandi. Si ha $\varphi(n) = (p-1)(q-1)$. L'algoritmo sfrutta il fatto che, conoscendo p e q , è facile calcolare $\varphi(n)$, ma senza conoscere la fattorizzazione di n , calcolare $\varphi(n)$ è computazionalmente intrattabile.

Se scegliamo un esponente e tale che $\gcd(e, \varphi(n)) = 1$, allora esiste un unico d tale che:

$$ed \equiv 1 \pmod{\varphi(n)}$$

Questo significa che $ed = 1 + k \cdot \varphi(n)$ per qualche intero k . Per il Teorema di Eulero, per ogni messaggio m coprimo con n :

$$m^{ed} = m^{1 + k \cdot \varphi(n)} = m \cdot (m^{\varphi(n)})^k \equiv m \cdot 1^k \equiv m \pmod{n}$$

Questo dimostra che cifrare con e e decifrare con d sono operazioni inverse, il che è la proprietà fondamentale di RSA.

1.3 Il Teorema Cinese del Resto

Il Teorema Cinese del Resto (TCR) è uno strumento fondamentale sia nella teoria dei numeri che nelle applicazioni crittografiche. Esso fornisce un metodo per risolvere sistemi di congruenze simultanee e stabilisce un isomorfismo algebrico cruciale.

Enunciato classico:

Siano $n_1, n_2, \dots, n_k$ interi positivi a due a due coprimi (cioè $\gcd(n_i, n_j) = 1$ per ogni $i \neq j$). Allora per qualunque scelta di interi $a_1, a_2, \dots, a_k$, il sistema di congruenze:

$$x \equiv a_1 \pmod{n_1} \quad x \equiv a_2 \pmod{n_2} \quad \dots \quad x \equiv a_k \pmod{n_k}$$

ha un'unica soluzione modulo $N = n_1 \cdot n_2 \cdot \dots \cdot n_k$.

Dimostrazione costruttiva:

La dimostrazione non solo stabilisce l'esistenza e l'unicità della soluzione, ma fornisce anche un metodo esplicito per costruirla.

Sia $N = n_1 \cdot n_2 \cdot \dots \cdot n_k$ il prodotto di tutti i moduli. Per ogni i , definiamo:

$$N_i = N / n_i = n_1 \cdot \dots \cdot n_{i-1} \cdot n_{i+1} \cdot \dots \cdot n_k$$

Osserviamo che $\gcd(N_i, n_i) = 1$, perché N_i è il prodotto di tutti gli n_j con $j \neq i$, e questi sono tutti coprimi con n_i per ipotesi.

Poiché $\gcd(N_i, n_i) = 1$, per il Teorema di Bézout (o per il Piccolo Teorema di Fermat generalizzato), esiste un inverso moltiplicativo M_i di N_i modulo n_i , cioè:

$$N_i \cdot M_i \equiv 1 \pmod{n_i}$$

Ora costruiamo la soluzione come:

$$x = a_1 N_1 M_1 + a_2 N_2 M_2 + \dots + a_k N_k M_k$$

Verifichiamo che questa x soddisfa tutte le congruenze. Consideriamo la congruenza modulo n_i :

$$x \equiv a_1 N_1 M_1 + a_2 N_2 M_2 + \dots + a_k N_k M_k \pmod{n_i}$$

Per ogni $j \neq i$, N_j contiene n_i come fattore (perché $N_j = N/n_j$ e N contiene tutti i moduli), quindi:

$$N_j M_j \equiv 0 \pmod{n_i} \text{ per ogni } j \neq i$$

Rimane solo il termine i -esimo:

$$x \equiv a_i N_i M_i \equiv a_i \cdot 1 \equiv a_i \pmod{n_i}$$

Questo dimostra che x soddisfa tutte le congruenze.

Unicità della soluzione:

Supponiamo che esistano due soluzioni x e y del sistema. Allora:

$$x \equiv a_i \pmod{n_i} \text{ e } y \equiv a_i \pmod{n_i} \text{ per ogni } i$$

Questo implica che $x - y \equiv 0 \pmod{n_i}$ per ogni i , cioè n_i divide $(x-y)$ per ogni i . Poiché gli n_i sono a due a due coprimi, il loro prodotto N divide $(x-y)$, quindi:

$$x \equiv y \pmod{N}$$

Questo dimostra l'unicità della soluzione modulo N .

Interpretazione algebrica:

Il TCR può essere formulato elegantemente usando la teoria degli anelli. Esso stabilisce un isomorfismo di anelli:

$$\mathbb{Z}/N\mathbb{Z} \cong \mathbb{Z}/n_1\mathbb{Z} \times \mathbb{Z}/n_2\mathbb{Z} \times \dots \times \mathbb{Z}/n_k\mathbb{Z}$$

dove $N = n_1 \cdot n_2 \cdot \dots \cdot n_k$ e gli n_i sono a due a due coprimi. L'isomorfismo è dato dalla mappa:

$$\varphi: \mathbb{Z}/N\mathbb{Z} \rightarrow \mathbb{Z}/n_1\mathbb{Z} \times \mathbb{Z}/n_2\mathbb{Z} \times \dots \times \mathbb{Z}/n_k\mathbb{Z} \quad \varphi(x \bmod N) = (x \bmod n_1, x \bmod n_2, \dots, x \bmod n_k)$$

Questo isomorfismo preserva sia l'addizione che la moltiplicazione, ed è fondamentale per comprendere la struttura algebrica degli anelli modulari.

Applicazioni crittografiche del TCR:

1. **Accelerazione delle operazioni in RSA:** L'operazione di decifratura in RSA richiede il calcolo di $m = c^d \bmod n$, dove $n = pq$. Usando il TCR, possiamo calcolare separatamente:

$$m_1 = c^d \bmod p \quad m_2 = c^d \bmod q$$

e poi ricostruire m usando il TCR. Poiché p e q sono circa $\sqrt{n}$, questa tecnica riduce notevolmente il tempo di calcolo. Infatti, la complessità dell'esponenziazione modulare è $O(\log^3 n)$, quindi lavorare con moduli di dimensione $\sqrt{n}$ invece che n riduce il tempo di circa un fattore 4.

Più precisamente, si calcolano:

$$d_1 = d \bmod (p-1) \quad d_2 = d \bmod (q-1)$$

e poi:

$$m_1 = c^{d_1} \bmod p \quad m_2 = c^{d_2} \bmod q$$

Per il TCR, esiste un'unica m con $0 \leq m < n$ tale che $m \equiv m_1 \pmod{p}$ e $m \equiv m_2 \pmod{q}$. Usando la formula costruttiva:

$$m = (m_1 \cdot q \cdot (q^{-1} \bmod p) + m_2 \cdot p \cdot (p^{-1} \bmod q)) \bmod n$$

Oppure, più comunemente, usando la forma:

$$h = (m_1 - m_2) \cdot (q^{-1} \bmod p) \bmod p \quad m = m_2 + h \cdot q$$

2. **Sistemi crittografici threshold:** Il TCR permette di dividere una chiave segreta in parti multiple in modo che sia necessario un certo numero minimo di parti per ricostruirla.
3. **Implementazioni resistenti agli attacchi side-channel:** Lavorare separatamente modulo p e q può rendere più difficili certi attacchi basati sull'analisi dei tempi o del consumo energetico.

1.4 La Ricerca di Numeri Primi e i Test di Primalità

La generazione e verifica di numeri primi grandi è essenziale per la crittografia moderna. Mentre esistono algoritmi deterministici per testare la primalità (come il test AKS del 2002), nella pratica si utilizzano spesso test probabilistici per la loro maggiore efficienza.

Il Test di Fermat:

Il test di primalità di Fermat si basa sul Piccolo Teorema di Fermat. Se n è primo e $\gcd(a,n) = 1$, allora $a^{(n-1)} \equiv 1 \pmod{n}$. Il test funziona così:

1. Scegliamo un intero a con $1 < a < n$
2. Calcoliamo $a^{(n-1)} \pmod{n}$
3. Se $a^{(n-1)} \not\equiv 1 \pmod{n}$, allora n è sicuramente composto
4. Se $a^{(n-1)} \equiv 1 \pmod{n}$, allora n è "probabilmente" primo

Un numero a per cui $a^{(n-1)} \equiv 1 \pmod{n}$ ma n è composto è chiamato "testimone di Fermat falso" o "pseudoprimo in base a ".

Numeri di Carmichael:

Purtroppo, esistono numeri composti n per cui $a^{(n-1)} \equiv 1 \pmod{n}$ per ogni a coprimo con n . Questi numeri sono chiamati numeri di Carmichael. Il più piccolo è $561 = 3 \times 11 \times 17$.

Un numero n è un numero di Carmichael se e solo se:

1. n è dispari
2. n è libero da quadrati (nessun primo compare con esponente > 1 nella fattorizzazione)
3. Per ogni primo p che divide n , si ha $(p-1)$ divide $(n-1)$

Il Test di Miller-Rabin:

Il test di Miller-Rabin migliora il test di Fermat sfruttando proprietà aggiuntive dei numeri primi. È uno dei test probabilistici più utilizzati nella pratica.

Sia $n > 2$ un numero dispari da testare. Scriviamo $n-1 = 2^s \cdot d$ dove d è dispari (estraiamo tutte le potenze di 2 da $n-1$).

Per un intero a con $1 < a < n$, definiamo a "testimone forte di compositezza" per n se:

1. $a^d \not\equiv 1 \pmod{n}$, E
2. $a^{(2^r \cdot d)} \not\equiv -1 \pmod{n}$ per ogni r con $0 \leq r < s$

Se n è primo, allora non esistono testimoni forti di compositezza per n . Se n è composto, allora almeno $3/4$ degli interi a con $1 < a < n$ sono testimoni forti di compositezza.

Algoritmo di Miller-Rabin:

Input: n (numero dispari > 2 da testare), k (numero di iterazioni)

Output: "composto" o "probabilmente primo"

1. Scrivi $n-1 = 2^s \cdot d$ con d dispari
2. Ripeti k volte:
 - a. Scegli a casualmente con $1 < a < n$
 - b. $x \leftarrow a^d \pmod{n}$
 - c. Se $x = 1$ o $x = n-1$, continua al prossimo a
 - d. Ripeti $s-1$ volte:

- $x \leftarrow x^2 \bmod n$
 - Se $x = n-1$, continua al prossimo a
 - e. Restituisci "composto"
3. Restituisci "probabilmente primo"

Analisi matematica del test di Miller-Rabin:

La correttezza del test si basa sulle seguenti proprietà dei numeri primi:

Proprietà 1: Se p è primo e $a^d \equiv 1 \pmod{p}$, allora il test passa (correttamente).

Proprietà 2: Se p è primo e $a^d \not\equiv 1 \pmod{p}$, consideriamo la sequenza:

$$a^d, a^{2d}, a^{4d}, \dots, a^{2^{s-1}d}, a^{2^s d} = a^{p-1}$$

Per il Piccolo Teorema di Fermat, l'ultimo termine è congruo a 1 modulo p . Se $x^2 \equiv 1 \pmod{p}$ per un primo p , allora $(x-1)(x+1) \equiv 0 \pmod{p}$, quindi $x \equiv \pm 1 \pmod{p}$. Questo significa che nella sequenza sopra, il primo elemento congruo a 1 deve essere preceduto da un elemento congruo a -1 (oppure il primo elemento stesso è 1). Quindi il test passa (correttamente).

Proprietà 3: Se n è composto, la probabilità che un a scelto casualmente non sia un testimone forte di compostezza è al massimo $1/4$.

Dimostrazione del bound $1/4$:

Questo è uno dei risultati più profondi nella teoria dei test di primalità. La dimostrazione completa richiede concetti avanzati di teoria dei gruppi.

Sia n dispari e composto, con $n-1 = 2^s \cdot d$. Definiamo:

$$L = \{a \in (\mathbb{Z}/n\mathbb{Z})^* : a \text{ non è un testimone forte di compostezza per } n\}$$

Vogliamo dimostrare che $|L| \leq \phi(n)/4$.

Caso 1: $n = p^k$ per un primo dispari p e $k \geq 2$.

In questo caso, consideriamo l'elemento $1 + p$. Si può dimostrare che $(1+p)^{p^{k-1}} \equiv 1 + p^k \equiv 1 \pmod{p^k}$, ma $(1+p)^{p^{k-2}} \not\equiv 1 \pmod{p^k}$ per $k \geq 2$. Inoltre, nessuna potenza intermedia è congrua a -1. Quindi $1+p$ è un testimone forte di compostezza, e questo implica che almeno metà degli elementi non sono in L .

Caso 2: n ha almeno due fattori primi distinti dispari.

Scriviamo $n = p_1^{a_1} \cdot \dots \cdot p_k^{a_k}$ con $k \geq 2$. Per il Teorema Cinese del Resto:

$$(\mathbb{Z}/n\mathbb{Z})^* \cong (\mathbb{Z}/p_1^{a_1}\mathbb{Z})^* \times \dots \times (\mathbb{Z}/p_k^{a_k}\mathbb{Z})^*$$

Un elemento $a \in L$ deve soddisfare $a^d \equiv 1 \pmod{n}$ oppure $a^{2^r d} \equiv -1 \pmod{n}$ per qualche r .

Si può dimostrare che se consideriamo l'insieme B di elementi che soddisfano queste condizioni, allora B è un sottogruppo proprio di $(\mathbb{Z}/n\mathbb{Z})^*$. Infatti, possiamo costruire un elemento che non appartiene a B modificando le componenti rispetto ai diversi fattori primi usando il TCR.

Per il Teorema di Lagrange, $|B|$ divide $\varphi(n)$, e poiché B è un sottogruppo proprio, $|B| \leq \varphi(n)/2$.

Con analisi più raffinata (considerando la struttura del gruppo e l'interazione tra le diverse componenti), si può dimostrare che in realtà $|B| \leq \varphi(n)/4$.

Garanzie probabilistiche:

Se n è composto e eseguiamo k iterazioni del test di Miller-Rabin con valori di a scelti indipendentemente e uniformemente a caso, la probabilità che n passi tutte le k iterazioni è al massimo $(1/4)^k = 4^{-k}$.

Per $k = 20$, la probabilità di errore è al massimo $4^{-20} \approx 10^{-12}$, che è accettabile per la maggior parte delle applicazioni crittografiche.

Per $k = 64$, la probabilità di errore è al massimo $4^{-64} \approx 10^{-38}$, che è estremamente sicura.

Versione deterministica di Miller-Rabin:

Esistono versioni deterministiche del test di Miller-Rabin sotto l'ipotesi di Riemann generalizzata (GRH). Se la GRH è vera, allora testando tutti i valori di a con $1 < a \leq 2(\ln n)^2$, possiamo determinare con certezza se n è primo.

Inoltre, per valori specifici di n , sono noti insiemi finiti di basi che garantiscono la correttezza. Ad esempio:

- Per $n < 2,047$, basta testare $a = 2$
- Per $n < 1,373,653$, basta testare $a \in \{2, 3\}$
- Per $n < 25,326,001$, basta testare $a \in \{2, 3, 5\}$
- Per $n < 3,215,031,751$, basta testare $a \in \{2, 3, 5, 7\}$
- Per $n < 2,152,302,898,747$, basta testare $a \in \{2, 3, 5, 7, 11\}$
- Per $n < 3,474,749,660,383$, basta testare $a \in \{2, 3, 5, 7, 11, 13\}$
- Per $n < 341,550,071,728,321$, basta testare $a \in \{2, 3, 5, 7, 11, 13, 17\}$

Questi risultati sono stati ottenuti attraverso ricerca computazionale esaustiva e forniscono garanzie deterministiche per intervalli praticamente rilevanti.

Il Test AKS (Agrawal-Kayal-Saxena)

Nel 2002, Agrawal, Kayal e Saxena hanno pubblicato il primo test di primalità deterministico con complessità polinomiale provata.

Idea chiave: Il test si basa sulla seguente generalizzazione del Piccolo Teorema di Fermat:

Teorema: n è primo se e solo se:

$$(x + a)^n \equiv x^n + a \pmod{n}$$

per ogni a coprimo con n , dove l'uguaglianza è tra polinomi in $\mathbb{Z}[x]$.

Problema: Verificare questa identità richiede tempo esponenziale perché il polinomio $(x + a)^n$ ha $n + 1$ termini.

Soluzione AKS: Verificare l'identità modulo un polinomio di grado piccolo $x^r - 1$:

$$(x + a)^n \equiv x^n + a \pmod{x^r - 1, n}$$

per un opportuno $r = O(\log^6 n)$ e un insieme limitato di valori di a .

Algoritmo AKS:

Input: intero $n > 1$

1. Se $n = a^b$ per $a \in \mathbb{N}$ e $b > 1$, output COMPOSTO
2. Trova il più piccolo r tale che $\text{ord}_r(n) > \log^2(n)$
3. Se $1 < \gcd(a, n) < n$ per qualche $a \leq r$, output COMPOSTO
4. Se $n \leq r$, output PRIMO
5. Per $a = 1$ fino a $\lfloor \sqrt[r]{\phi(r) \cdot \log(n)} \rfloor$:
 - Se $(x + a)^n \not\equiv x^n + a \pmod{x^r - 1, n}$, output COMPOSTO
6. Output PRIMO

Complessità: $O(\log^{12+\epsilon}(n))$ nel paper originale, migliorata a $O(\log^6(n))$ con raffinamenti successivi.

Significato teorico: Prima dimostrazione che $\text{PRIMES} \in \text{P}$ (problemi decidibili in tempo polinomiale deterministico).

Uso pratico: Nonostante la complessità polinomiale garantita, AKS è più lento di Miller-Rabin probabilistico per dimensioni pratiche ($< 10^6$ cifre), quindi Miller-Rabin rimane preferito nella pratica.

Parte II: Strutture Algebriche Avanzate e Curve Ellittiche

2. Teoria dei Gruppi e Strutture Algebriche

2.1 Gruppi Ciclici e Generatori

Un gruppo ciclico è un gruppo che può essere generato da un singolo elemento.

Formalmente, un gruppo G è ciclico se esiste un elemento $g \in G$ tale che ogni elemento di G può essere scritto come g^k per qualche intero k .

Definizione rigorosa:

Sia $(G, \cdot)$ un gruppo. Un elemento $g \in G$ è chiamato generatore di G se:

$$G = \langle g \rangle = \{g^k : k \in \mathbb{Z}\}$$

dove g^k è definito come:

- $g^0 = e$ (elemento neutro)
- $g^k = g \cdot g \cdot \dots \cdot g$ (k volte) per $k > 0$
- $g^{-k} = (g^{-1})^k$ per $k < 0$

Gruppi ciclici finiti:

Se G è un gruppo ciclico finito di ordine n (cioè $|G| = n$), allora:

$$G = \{e, g, g^2, g^3, \dots, g^{(n-1)}\}$$

e si ha $g^n = e$. Inoltre, ogni elemento di G ha la forma g^k con $0 \leq k < n$.

Teorema fondamentale sui gruppi ciclici:

Ogni gruppo ciclico finito di ordine n è isomorfo al gruppo additivo $(\mathbb{Z}/n\mathbb{Z}, +)$. L'isomorfismo è dato da:

$$\varphi: \mathbb{Z}/n\mathbb{Z} \rightarrow G \quad \varphi(k) = g^k$$

dove g è un generatore di G .

Sottogruppi di gruppi ciclici:

Un risultato fondamentale afferma che ogni sottogruppo di un gruppo ciclico è ciclico. Più precisamente:

Teorema: Sia $G = \langle g \rangle$ un gruppo ciclico di ordine n , e sia H un sottogruppo di G . Allora:

1. H è ciclico
2. Se $|H| = d$, allora d divide n
3. $H = \langle g^{(n/d)} \rangle$

Dimostrazione:

Sia k il più piccolo intero positivo tale che $g^k \in H$. Affermiamo che $H = \langle g^k \rangle$.

Sia $h \in H$ un elemento arbitrario. Poiché $h \in G = \langle g \rangle$, esiste un intero m tale che $h = g^m$. Dividendo m per k , otteniamo:

$$m = qk + r \text{ con } 0 \leq r < k$$

Quindi:

$$h = g^m = g^{(qk+r)} = (g^k)^q \cdot g^r$$

Poiché $g^k \in H$ e H è un gruppo, $(g^k)^q \in H$. Quindi:

$$g^r = h \cdot ((g^k)^q)^{-1} \in H$$

Ma $0 \leq r < k$, e k è il più piccolo intero positivo tale che $g^k \in H$. Quindi $r = 0$, il che implica $h = (g^k)^q \in \langle g^k \rangle$.

Abbiamo dimostrato che $H \subseteq \langle g^k \rangle$. L'inclusione opposta è ovvia, quindi $H = \langle g^k \rangle$.

Ora, per il Teorema di Lagrange, $|H|$ divide $|G| = n$. Sia $d = |H|$. Poiché $H = \langle g^k \rangle$ e $|H| = d$, l'ordine di g^k è d . Ma l'ordine di g^k in G è $n/\gcd(n,k)$. Quindi:

$$d = n/\gcd(n,k)$$

Questo implica che $\gcd(n,k) = n/d$, quindi $k = n/d$ (prendendo il valore minimo di k). Quindi $H = \langle g^{(n/d)} \rangle$.

Numero di generatori:

Un gruppo ciclico di ordine n ha esattamente $\varphi(n)$ generatori, dove φ è la funzione di Eulero. Infatti, g^k è un generatore di $G = \langle g \rangle$ se e solo se $\gcd(k,n) = 1$.

Dimostrazione:

L'ordine di g^k è $n/\gcd(n,k)$. Quindi g^k genera l'intero gruppo G se e solo se l'ordine di g^k è n , cioè se e solo se $\gcd(n,k) = 1$. Gli interi k con $0 \leq k < n$ e $\gcd(k,n) = 1$ sono esattamente $\varphi(n)$.

2.2 Il Gruppo Moltiplicativo $(\mathbb{Z}/p\mathbb{Z})^*$

Per un primo p , il gruppo moltiplicativo $(\mathbb{Z}/p\mathbb{Z})^* = \{1, 2, \dots, p-1\}$ con l'operazione di moltiplicazione modulo p è un gruppo ciclico di ordine $p-1$.

Teorema fondamentale:

Per ogni primo p , il gruppo $(\mathbb{Z}/p\mathbb{Z})^*$ è ciclico.

Dimostrazione (sketch):

La dimostrazione completa richiede risultati profondi sulla struttura dei gruppi abeliani finiti. L'idea chiave è la seguente:

Per ogni divisore d di $p-1$, il numero di elementi in $(\mathbb{Z}/p\mathbb{Z})^*$ di ordine d è al massimo d (perché sono le radici del polinomio $x^d - 1 = 0$ in $\mathbb{Z}/p\mathbb{Z}$, e un polinomio di grado d ha al massimo d radici in un campo).

D'altra parte, se $(\mathbb{Z}/p\mathbb{Z})^*$ è ciclico, allora ha esattamente $\varphi(d)$ elementi di ordine d per ogni divisore d di $p-1$.

Si può dimostrare che:

$$\sum_{d|n} \phi(d) = n$$

Quindi la somma del numero di elementi di ordine d per tutti i divisori d di $p-1$ è esattamente $p-1$, che è l'ordine di $(\mathbb{Z}/p\mathbb{Z})^*$. Questo è possibile solo se per ogni divisore d ci sono esattamente $\phi(d)$ elementi di ordine d , il che implica che il gruppo è ciclico.

Generatori primitivi:

Un generatore del gruppo $(\mathbb{Z}/p\mathbb{Z})^*$ è chiamato radice primitiva modulo p . Per ogni primo p , esistono $\phi(p-1)$ radici primitive modulo p .

Esempio: Per $p = 7$, $(\mathbb{Z}/7\mathbb{Z})^* = \{1, 2, 3, 4, 5, 6\}$. Verifichiamo che 3 è una radice primitiva:

$$3^1 \equiv 3 \pmod{7} \quad 3^2 \equiv 2 \pmod{7} \quad 3^3 \equiv 6 \pmod{7} \quad 3^4 \equiv 4 \pmod{7} \quad 3^5 \equiv 5 \pmod{7} \quad 3^6 \equiv 1 \pmod{7}$$

Quindi 3 genera l'intero gruppo. Il numero di radici primitive modulo 7 è $\phi(6) = 2$. L'altra radice primitiva è 5.

Trovare radici primitive:

Non esiste un metodo efficiente noto per trovare deterministicamente una radice primitiva modulo p . Tuttavia, se $p-1$ ha pochi fattori primi distinti, possiamo verificare efficientemente se un dato elemento g è una radice primitiva:

g è una radice primitiva modulo p se e solo se $g^{(p-1)/q} \not\equiv 1 \pmod{p}$ per ogni fattore primo q di $p-1$.

Dimostrazione:

L'ordine di g divide $p-1$. Se $g^{(p-1)/q} \equiv 1 \pmod{p}$ per qualche fattore primo q di $p-1$, allora l'ordine di g divide $(p-1)/q$, quindi è strettamente minore di $p-1$, quindi g non è una radice primitiva.

Viceversa, se l'ordine di g è $d < p-1$, allora d divide $p-1$, quindi esiste un fattore primo q di $p-1$ tale che d divide $(p-1)/q$. Ma allora $g^{(p-1)/q} = (g^d)^{(p-1)/(qd)} \equiv 1 \pmod{p}$.

Algoritmo per trovare una radice primitiva:

Input: primo p

Output: una radice primitiva g modulo p

1. Fattorizza $p-1 = q_1^{a_1} \cdot \dots \cdot q_k^{a_k}$
2. Ripeti:
 - a. Scegli g casualmente con $2 \leq g \leq p-1$
 - b. $\text{test_ok} \leftarrow \text{true}$
 - c. Per ogni i da 1 a k :
 - Se $g^{(p-1)/q_i} \equiv 1 \pmod{p}$:

```

* test_ok ← false
* esci dal ciclo
d. Se test_ok = true:
  - Restituisci g

```

Sotto ragionevoli assunzioni euristiche, questo algoritmo trova una radice primitiva dopo $O(\log p)$ tentativi in media.

2.3 Il Problema del Logaritmo Discreto

Il problema del logaritmo discreto (DLP) è uno dei problemi matematici fondamentali su cui si basa la sicurezza di molti sistemi crittografici.

Definizione formale:

Sia G un gruppo ciclico di ordine n , e sia g un generatore di G . Dato un elemento $h \in G$, il problema del logaritmo discreto consiste nel trovare l'intero x con $0 \leq x < n$ tale che:

$$g^x = h$$

L'intero x è chiamato logaritmo discreto di h in base g , e si scrive $x = \log_g(h)$.

Il DLP nel gruppo moltiplicativo:

Nel contesto di $(\mathbb{Z}/p\mathbb{Z})^*$, dove p è primo e g è una radice primitiva modulo p , il problema diventa:

Dati p , g , e $h = g^x \bmod p$, trovare x .

Difficoltà computazionale:

Per gruppi ben scelti, non si conoscono algoritmi efficienti per risolvere il DLP. I migliori algoritmi noti hanno complessità sub-esponenziale o esponenziale nella dimensione del gruppo:

1. **Algoritmo Baby-Step Giant-Step**: Complessità $O(\sqrt{n} \cdot \log n)$ in tempo e $O(\sqrt{n})$ in spazio, dove n è l'ordine del gruppo.
2. **Algoritmo di Pollard's rho**: Complessità $O(\sqrt{n})$ in tempo e $O(1)$ in spazio.
3. **Index Calculus** (per $(\mathbb{Z}/p\mathbb{Z})^*$): Complessità sub-esponenziale $L_p[1/2, c]$ dove:

$$L_p[\alpha, c] = \exp((c + o(1))(\ln p)^\alpha (\ln \ln p)^{(1-\alpha)})$$

Per lo stato dell'arte attuale, $\alpha \approx 1/3$ e $c \approx (64/9)^{(1/3)}$.

Algoritmo Baby-Step Giant-Step:

Questo è un algoritmo classico di meet-in-the-middle per il DLP.

Sia n l'ordine del gruppo, e poniamo $m = \lceil \sqrt{n} \rceil$. Scriviamo $x = qm + r$ dove $0 \leq r < m$ e $0 \leq q < m$.

$$\text{Allora: } h = g^x = g^{(qm+r)} = (g^m)^q \cdot g^r$$

$$\text{Quindi: } h \cdot (g^{(-r)}) = (g^m)^q$$

L'algoritmo procede così:

Baby steps:

1. Calcola e memorizza (r, g^r) per $r = 0, 1, \dots, m-1$ in una tabella hash indicizzata da g^r

Giant steps:

2. Calcola $\gamma = g^m$
3. Inizializza $c = h$
4. Per $q = 0, 1, \dots, m-1$:
 - a. Cerca c nella tabella
 - b. Se $c = g^r$ per qualche r :
 - Restituisci $x = qm + r$
 - c. $c \leftarrow c \cdot \gamma$
5. Errore (non dovrebbe accadere se h è nel gruppo generato da g)

Complessità: $O(\sqrt{n})$ calcoli di gruppo e $O(\sqrt{n})$ spazio per la tabella.

Algoritmo Pollard's rho:

L'algoritmo di Pollard utilizza una funzione pseudo-casuale per trovare una "collisione" che rivela il logaritmo discreto. È più efficiente in termini di spazio rispetto a Baby-Step Giant-Step.

L'idea è di costruire una sequenza pseudo-casuale:

$$x_i, a_i, b_i$$

dove $x_i = g^{a_i} \cdot h^{b_i}$, partendo da $(x_0, a_0, b_0) = (1, 0, 0)$.

La funzione di iterazione divide lo spazio in tre regioni e applica una regola diversa in ciascuna:

$$\text{Se } x_i \in S_1: x_{i+1} = h \cdot x_i, \quad a_{i+1} = a_i, \quad b_{i+1} = b_i + 1$$

$$\text{Se } x_i \in S_2: x_{i+1} = x_i^2, \quad a_{i+1} = 2a_i, \quad b_{i+1} = 2b_i$$

$$\text{Se } x_i \in S_3: x_{i+1} = g \cdot x_i, \quad a_{i+1} = a_i + 1, \quad b_{i+1} = b_i$$

Per il paradosso del compleanno, dopo circa $\sqrt{(\pi n/2)}$ iterazioni, ci aspettiamo una collisione $x_i = x_j$ per qualche $i \neq j$.

Quando questo accade:

$$g^{a_i} \cdot h^{b_i} = g^{a_j} \cdot h^{b_j}$$

Se $b_i \neq b_j \pmod{n}$, possiamo risolvere per $\log_g(h)$:

$$h^{(b_i - b_j)} = g^{(a_j - a_i)} \quad h = g^{((a_j - a_i) / (b_i - b_j))}$$

quindi:

$$\log_g(h) = (a_j - a_i) \cdot (b_i - b_j)^{-1} \pmod{n}$$

Rilevazione della collisione (algoritmo di Floyd):

Per rilevare la collisione senza memorizzare tutti i valori, usiamo due puntatori che si muovono a velocità diverse:

1. Inizializza $(x, a, b) = (1, 0, 0)$
2. Inizializza $(X, A, B) = (1, 0, 0)$
3. Ripeti:
 - a. $(x, a, b) \leftarrow \text{itera}((x, a, b))$ una volta
 - b. $(X, A, B) \leftarrow \text{itera}((X, A, B))$ due volte
 - c. Se $x = X$:
 - Se $b \equiv B \pmod{n}$, riprova con valori iniziali diversi
 - Altrimenti calcola $\log_g(h) = (A-a)(b-B)^{-1} \pmod{n}$
 - Restituisci il risultato

Complessità: $O(\sqrt{n})$ operazioni di gruppo con $O(1)$ spazio.

2.4 Il Problema Diffie-Hellman Computazionale

Il problema Diffie-Hellman computazionale (CDH) è strettamente legato al DLP ma potenzialmente più debole.

Definizione:

Sia G un gruppo ciclico di ordine n generato da g . Dati g^a e g^b per interi a e b sconosciuti, calcolare g^{ab} .

Relazione con il DLP:

Se possiamo risolvere il DLP, allora possiamo risolvere il CDH: dati g^a e g^b , calcoliamo $a = \log_g(g^a)$, e poi $g^{ab} = (g^b)^a$.

Non è noto se la riduzione opposta sia possibile, cioè se risolvere il CDH implichi poter risolvere il DLP. È congetturato che il CDH sia strettamente più facile del DLP, ma non è mai stato dimostrato.

Il problema Diffie-Hellman decisionale (DDH):

Una variante ancora più debole è il problema DDH: dati g^a , g^b , e g^c , decidere se $c = ab \pmod{n}$ oppure c è casuale.

Il DDH è più facile del CDH, che è più facile del DLP:

$$\text{DDH} \leq \text{CDH} \leq \text{DLP}$$

Alcuni gruppi in cui il DLP è ritenuto difficile hanno comunque un DDH facile. Ad esempio, su curve ellittiche con accoppiamenti bilineari (pairing), il DDH è risolvibile in tempo polinomiale usando il pairing, ma il DLP è ancora considerato difficile.

3. Curve Ellittiche

Le curve ellittiche forniscono gruppi in cui il DLP sembra particolarmente difficile, permettendo l'uso di chiavi più corte rispetto a $(\mathbb{Z}/p\mathbb{Z})^*$ per lo stesso livello di sicurezza.

3.1 Definizione e Geometria delle Curve Ellittiche

Definizione algebrica:

Una curva ellittica E su un campo K (di caratteristica $\neq 2, 3$) è definita dall'equazione di Weierstrass:

$$y^2 = x^3 + ax + b$$

dove $a, b \in K$ soddisfano la condizione di non singolarità:

$$4a^3 + 27b^2 \neq 0$$

Questa condizione garantisce che la curva non abbia punti singolari (cuspidi o auto-intersezioni).

Punti sulla curva:

L'insieme dei punti della curva ellittica è:

$$E(K) = \{(x, y) \in K \times K : y^2 = x^3 + ax + b\} \cup \{O\}$$

dove O è un "punto all'infinito" che funge da elemento neutro per l'operazione di gruppo.

Interpretazione geometrica:

Su $\mathbb{R}$, una curva ellittica può essere visualizzata come una curva simmetrica rispetto all'asse x . La condizione $4a^3 + 27b^2 \neq 0$ garantisce che il polinomio cubico $x^3 + ax + b$ non abbia radici multiple, quindi la curva non ha punti singolari.

Ad esempio, la curva $y^2 = x^3 - x$ ha tre intersezioni con l'asse x (in $x = -1, 0, 1$) e due componenti connesse su $\mathbb{R}$.

3.2 La Legge di Gruppo su Curve Ellittiche

La caratteristica più notevole delle curve ellittiche è che l'insieme $E(K)$ può essere dotato di una struttura di gruppo abeliano.

Definizione geometrica dell'addizione:

Siano P e Q due punti su $E(K)$. Per definire $P + Q$:

1. **Se $P = O$:** $P + Q = Q$ (O è l'elemento neutro)
2. **Se $Q = O$:** $P + Q = P$
3. **Se $P \neq Q$ e $P \neq -Q$:** Traccia la retta passante per P e Q . Questa retta interseca la curva in un terzo punto R' . Definisci $P + Q$ come la riflessione di R' rispetto all'asse x .
4. **Se $P = Q$:** Traccia la tangente alla curva in P . Questa interseca la curva in un punto R' . Definisci $2P = P + P$ come la riflessione di R' rispetto all'asse x .
5. **Se $Q = -P$** (cioè P e Q hanno la stessa x ma y opposte): $P + Q = O$

Formule algebriche esplicite:

Siano $P = (x_1, y_1)$ e $Q = (x_2, y_2)$ punti su $E: y^2 = x^3 + ax + b$.

Caso $P \neq Q$:

Se $x_1 \neq x_2$, la pendenza della retta passante per P e Q è:

$$\lambda = (y_2 - y_1) / (x_2 - x_1)$$

Il terzo punto di intersezione $R = (x_3, y_3)$ soddisfa:

$$x_3 = \lambda^2 - x_1 - x_2 \quad y_3 = \lambda(x_1 - x_3) - y_1$$

Quindi $P + Q = (x_3, -y_3)$.

Caso $P = Q$ (raddoppio):

La pendenza della tangente in $P = (x_1, y_1)$ è:

$$\lambda = (3x_1^2 + a) / (2y_1)$$

(assumendo $y_1 \neq 0$, altrimenti $2P = O$)

Il punto $2P = (x_3, y_3)$ è dato da:

$$x_3 = \lambda^2 - 2x_1 \quad y_3 = \lambda(x_1 - x_3) - y_1$$

Quindi $2P = (x_3, -y_3)$.

Caso speciale $P = -Q$:

Se $P = (x, y)$ e $Q = (x, -y)$, allora $P + Q = O$.

Inverso:

L'inverso di $P = (x, y)$ è $-P = (x, -y)$.

Dimostrazione della associatività:

La proprietà più difficile da verificare è l'associatività: $(P + Q) + R = P + (Q + R)$ per tutti i punti P, Q, R .

La dimostrazione geometrica usa il Teorema di Pappus-Pascal dalla geometria proiettiva. La dimostrazione algebrica richiede calcoli laboriosi ma diretti, verificando che le formule esplicite soddisfano l'associatività.

Un approccio più elegante usa la teoria delle funzioni razionali sulla curva e il teorema sui divisori, che fornisce una dimostrazione più concettuale dell'associatività.

3.3 Curve Ellittiche su Campi Finiti

Per applicazioni crittografiche, consideriamo curve ellittiche su campi finiti $\mathbb{F}_p$ dove p è un primo grande (tipicamente $p > 2^{160}$).

Definizione:

Una curva ellittica su $\mathbb{F}_p$ è data da:

$$E(\mathbb{F}_p): y^2 \equiv x^3 + ax + b \pmod{p}$$

dove $a, b \in \mathbb{F}_p$ soddisfano $4a^3 + 27b^2 \not\equiv 0 \pmod{p}$.

Numero di punti (Teorema di Hasse):

Il numero di punti $|E(\mathbb{F}_p)|$ (incluso O) soddisfa:

$$|E(\mathbb{F}_p)| = p + 1 - t$$

dove $|t| \leq 2\sqrt{p}$. Il valore t è chiamato traccia di Frobenius.

Questo teorema, dovuto a Helmut Hasse, fornisce un bound stretto sul numero di punti. La dimostrazione usa proprietà profonde dell'endomorfismo di Frobenius e la teoria degli zeta-funzioni.

Algoritmi per contare i punti:

1. **Metodo ingenuo:** $O(p)$ operazioni, troppo lento per p grande.
2. **Algoritmo di Schoof:** $O(\log^8 p)$ operazioni, pratico per p moderatamente grande.

3. **Algoritmo SEA (Schoof-Elkies-Atkin):** $O(\log^6 p)$ operazioni, miglioramento dell'algoritmo di Schoof.
4. **Algoritmo baby-step giant-step** (per casi speciali): $O(p^{1/4})$ operazioni.

Struttura del gruppo $E(\mathbb{F}_p)$:

Per il teorema della struttura dei gruppi abeliani finiti:

$$E(\mathbb{F}_p) \cong \mathbb{Z}/n_1\mathbb{Z} \times \mathbb{Z}/n_2\mathbb{Z}$$

dove n_2 divide n_1 e $n_1 n_2 = |E(\mathbb{F}_p)|$.

Nella maggior parte dei casi pratici per la crittografia, scegliamo curve tali che $E(\mathbb{F}_p)$ sia ciclico o quasi ciclico (cioè n_2 piccolo, tipicamente $n_2 \in \{1, 2, 3, 4\}$).

3.4 Il Problema del Logaritmo Discreto su Curve Ellittiche (ECDLP)

Definizione:

Data una curva ellittica E su $\mathbb{F}_p$, un punto base $G \in E(\mathbb{F}_p)$ di ordine n , e un punto $Q \in \langle G \rangle$, trovare l'intero k con $0 \leq k < n$ tale che:

$$Q = kG = G + G + \dots + G \text{ (k volte)}$$

Difficoltà computazionale:

Per curve ellittiche ben scelte, i migliori algoritmi noti per l'ECDLP sono gli algoritmi generici (che funzionano su qualunque gruppo):

1. **Baby-Step Giant-Step:** $O(\sqrt{n})$ operazioni
2. **Pollard's rho:** $O(\sqrt{n})$ operazioni
3. **Pollard's lambda** (per intervalli limitati): $O(\sqrt{w})$ dove w è l'ampiezza dell'intervallo

Crucialmente, gli algoritmi sub-esponenziali come l'index calculus che funzionano su $(\mathbb{Z}/p\mathbb{Z})^*$ **non sono noti per curve ellittiche generiche.**

Implicazioni per la sicurezza:

Questa resistenza agli algoritmi sub-esponenziali significa che le chiavi ECC (Elliptic Curve Cryptography) possono essere molto più corte delle chiavi RSA o DH per lo stesso livello di sicurezza:

- 160 bit ECC $\approx$ 1024 bit RSA
- 256 bit ECC $\approx$ 3072 bit RSA
- 384 bit ECC $\approx$ 7680 bit RSA
- 521 bit ECC $\approx$ 15360 bit RSA

Curve deboli da evitare:

Non tutte le curve ellittiche sono sicure. Esistono classi di curve che sono vulnerabili:

1. **Curve anomale** (o curve tracciate): Curve su $\mathbb{F}_p$ con $|E(\mathbb{F}_p)| = p$. Su queste curve, l'ECDLP può essere risolto in tempo polinomiale usando il "sollevamento p-adico".
2. **Curve supersingolari**: Curve per cui $|E(\mathbb{F}_p)| = p + 1$. L'ECDLP può essere ridotto al DLP in un campo di estensione piccolo $\mathbb{F}_{p^k}$ con k piccolo (tipicamente $k \leq 6$), dove è applicabile l'index calculus.
3. **Curve con sottogruppo piccolo**: Se il gruppo $E(\mathbb{F}_p)$ ha un sottogruppo piccolo, informazioni sul logaritmo discreto possono trapelare attraverso il Pohlig-Hellman attack.

Curve standard e sicure:

Per evitare vulnerabilità, si utilizzano curve standard verificate:

- **NIST curves**: P-192, P-224, P-256, P-384, P-521
- **Curve25519 e Curve448**: Curve di Montgomery ottimizzate per prestazioni e sicurezza
- **secp256k1**: Usata in Bitcoin

Queste curve sono state accuratamente analizzate e si ritiene non abbiano strutture speciali sfruttabili.

3.5 Accoppiamenti Bilineari (Pairings)

Gli accoppiamenti bilineari su curve ellittiche sono funzioni speciali che mappano coppie di punti su curve ellittiche in elementi di un campo finito, mantenendo proprietà bilineari.

Definizione:

Sia E una curva ellittica su $\mathbb{F}_q$, e siano G_1, G_2 sottogruppi di ordine primo r di $E(\mathbb{F}_q)$ e $E(\mathbb{F}_{q^k})$ rispettivamente (dove k è chiamato embedding degree). Un accoppiamento bilineare è una funzione:

$$e: G_1 \times G_2 \rightarrow \mathbb{F}_{q^k}^*$$

con le seguenti proprietà:

1. **Bilinearità**: Per tutti $P, P' \in G_1, Q, Q' \in G_2$, e $a, b \in \mathbb{Z}$:

$$e(aP, bQ) = e(P, Q)^{ab} \quad e(P + P', Q) = e(P, Q) \cdot e(P', Q) \quad e(P, Q + Q') = e(P, Q) \cdot e(P, Q')$$

2. **Non-degeneratezza**: Se $e(P, Q) = 1$ per tutti $Q \in G_2$, allora $P = O$
3. **Calcolabilità efficiente**: Esiste un algoritmo polinomiale per calcolare $e(P, Q)$

Il Pairing di Weil:

Il pairing di Weil w_r è definito usando divisori e funzioni razionali sulla curva. Per punti $P \in E[r]$ e $Q \in E(\mathbb{F}_{(q^k)})$ dove r divide $|E(\mathbb{F}_{(q^k)})|$:

$$w_r(P, Q) = f_P(Q + S) / f_P(S) / f_Q(P + T) / f_Q(T)$$

dove f_P e f_Q sono funzioni razionali con divisori $\text{div}(f_P) = r[P] - r[O]$ e $\text{div}(f_Q) = r[Q] - r[O]$, e S, T sono punti ausiliari scelti opportunamente.

Il Pairing di Tate:

Il pairing di Tate è più efficiente da calcolare. Per $P \in E[r]$ e $Q \in E(\mathbb{F}_{(q^k)})$:

$$t_r(P, Q) = f_P(Q)^{(q^k - 1)/r}$$

dove f_P è una funzione razionale con divisore $\text{div}(f_P) = r[P] - r[O]$.

Algoritmo di Miller:

L'algoritmo di Miller calcola efficientemente $f_P(Q)$ per il pairing:

Input: punto P di ordine r , punto Q , intero r

Output: $f_P(Q)$ dove $\text{div}(f_P) = r[P] - r[O]$

Scrivi r in binario: $r = \sum r_i \cdot 2^i$

$T \leftarrow P$

$f \leftarrow 1$

Per i da $\ell-2$ fino a 0:

Sia $l_{T,T}$ la retta tangente in T

Sia v_{2T} la retta verticale per $2T$

$f \leftarrow f^2 \cdot l_{T,T}(Q) / v_{2T}(Q)$

$T \leftarrow 2T$

Se $r_i = 1$:

Sia $l_{T,P}$ la retta per T e P

Sia $v_{(T+P)}$ la retta verticale per $T+P$

$f \leftarrow f \cdot l_{T,P}(Q) / v_{(T+P)}(Q)$

$T \leftarrow T + P$

Restituisci f

Complessità: $O(\log r)$ operazioni nella curva ellittica, dove ogni operazione richiede valutazioni di rette in Q .

Proprietà matematiche degli accoppiamenti

1. Bilinearità completa:

Per tutti $P, P' \in G_1$, $Q, Q' \in G_2$, e scalari $a, b \in \mathbb{Z}$:

$$\begin{aligned}
e(aP, bQ) &= e(P, Q)^{ab} \\
e(P + P', Q) &= e(P, Q) \cdot e(P', Q) \\
e(P, Q + Q') &= e(P, Q) \cdot e(P, Q')
\end{aligned}$$

2. Non-degeneratezza:

Se $G_1 = \langle P \rangle$ e $G_2 = \langle Q \rangle$, allora $e(P, Q)$ è un generatore di un sottogruppo di ordine r in $F^*(q^k)$.

3. Calcolabilità:

L'algoritmo di Miller calcola $e(P, Q)$ in tempo $O(\log r)$ con costanti piccole.

Applicazioni crittografiche dei pairing

1. Identity-Based Encryption (IBE):

Schema di Boneh-Franklin (2001):

- Setup: Master key s , pubblica $P_{\text{pub}} = sP$
- Extract: Per identità ID , chiave privata $d_{ID} = sH(ID)$ dove $H: \{0,1\}^* \rightarrow G_1$
- Encrypt: $c = (rP, M \oplus H'(e(H(ID), P_{\text{pub}})^r))$
- Decrypt: $M = c_2 \oplus H'(e(d_{ID}, c_1))$

Correttezza:

$$e(d_{ID}, c_1) = e(sH(ID), rP) = e(H(ID), P)^{sr} = e(H(ID), sP)^r = e(H(ID), P_{\text{pub}})^r$$

2. Short Signatures (BLS):

Schema Boneh-Lynn-Shacham:

- Chiave privata: $x \in \mathbb{Z}_r$
- Chiave pubblica: $X = xP_2$
- Firma di m : $\sigma = xH(m)$ dove $H: \{0,1\}^* \rightarrow G_1$
- Verifica: $e(\sigma, P_2) = e(H(m), X)$

Correttezza:

$$e(\sigma, P_2) = e(xH(m), P_2) = e(H(m), P_2)^x = e(H(m), xP_2) = e(H(m), X)$$

Lunghezza firma: Solo un elemento di G_1 (tipicamente 48 byte per BLS12-381), molto più compatta di ECDSA (64 byte).

3. Tripartite Diffie-Hellman (Joux):

Tre parti A, B, C possono stabilire una chiave condivisa in un solo round:

- A sceglie a , pubblica aP
- B sceglie b , pubblica bP
- C sceglie c , pubblica cP
- Chiave condivisa: $K = e(P, P)^{abc}$

Ciascuna parte può calcolare K :

- A: $e(bP, cP)^a = e(P, P)^{abc}$
- B: $e(aP, cP)^b = e(P, P)^{abc}$
- C: $e(aP, bP)^c = e(P, P)^{abc}$

Curve pairing-friendly

Non tutte le curve permettono accoppiamenti efficienti. Serve un embedding degree k piccolo ma non troppo piccolo:

- k troppo piccolo $\rightarrow$ DDH diventa facile
- k troppo grande $\rightarrow$ pairing inefficiente

Curve standard:

BN (Barreto-Naehrig) curves:

$E: y^2 = x^3 + b$ su F_p

$p = 36u^4 + 36u^3 + 24u^2 + 6u + 1$

$n = 36u^4 + 36u^3 + 18u^2 + 6u + 1$

$k = 12$

BLS (Barreto-Lynn-Scott) curves:

BLS12-381: $k = 12$, $p \approx 2^{381}$, sicurezza ~ 128 bit

BLS24: $k = 24$, maggiore sicurezza

Sicurezza dei pairing

DDH è facile: Dato g, g^a, g^b, g^c , decidere se $c = ab$:

Calcola $e(g, g^c)$ e $e(g^a, g^b)$

Se uguali, allora $c = ab$

Ma CDH e DLP rimangono difficili: Dati g, g^a, g^b , calcolare g^{ab} o trovare a non è più facile grazie al pairing.

Attacchi su pairing:

- MOV attack: riduce ECDLP a DLP in $F_{(q^k)}$
- Funziona solo se k è piccolo

- Per curve pairing-friendly, k è scelto per bilanciare efficienza e sicurezza

Parte III: Algoritmi Crittografici Fondamentali - RSA e Diffie-Hellman

4. L'Algoritmo RSA

L'algoritmo RSA (Rivest-Shamir-Adleman, 1977) è stato il primo sistema crittografico a chiave pubblica pratico e rimane uno dei più utilizzati. La sua sicurezza si basa sulla difficoltà computazionale della fattorizzazione di numeri interi grandi.

4.1 Fondamenti Matematici di RSA

Il problema della fattorizzazione:

Dato un intero n che è il prodotto di due primi distinti grandi p e q , trovare p e q è computazionalmente difficile quando n è sufficientemente grande (tipicamente $n > 2^{2048}$).

I migliori algoritmi classici conosciuti per la fattorizzazione sono sub-esponenziali:

- **Crivello Quadratico (QS)**: Complessità $L_n[1/2, 1]$ dove $L_n[\alpha, c] = \exp((c + o(1))(\ln n)^\alpha (\ln \ln n)^{(1-\alpha)})$
- **Crivello Generale del Corpo dei Numeri (GNFS)**: Complessità $L_n[1/3, (64/9)^{1/3}] \approx L_n[1/3, 1.923]$

Per n di 2048 bit, la fattorizzazione con GNFS richiederebbe circa 2^{140} operazioni, che è considerato sicuro contro attacchi pratici.

Costruzione matematica di RSA:

Il sistema RSA sfrutta la seguente proprietà algebrica fondamentale, che deriva direttamente dal Teorema di Eulero:

Teorema fondamentale di RSA: Sia $n = pq$ dove p e q sono primi distinti. Sia $\phi(n) = (p-1)(q-1)$. Se $e \cdot d \equiv 1 \pmod{\phi(n)}$, allora per ogni intero m con $0 \leq m < n$:

$$m^{ed} \equiv m \pmod{n}$$

Dimostrazione rigorosa:

Dobbiamo dimostrare che $m^{ed} \equiv m \pmod{n}$, cioè che n divide $m^{ed} - m$.

Poiché $n = pq$ e $\gcd(p, q) = 1$, per il Teorema Cinese del Resto è sufficiente dimostrare che:

$$m^{ed} \equiv m \pmod{p} \text{ e } m^{ed} \equiv m \pmod{q}$$

Dimostriamo la congruenza modulo p (la dimostrazione modulo q è identica).

Poiché $ed \equiv 1 \pmod{\varphi(n)}$ e $\varphi(n) = (p-1)(q-1)$, esiste un intero k tale che:

$$ed = 1 + k(p-1)(q-1)$$

Quindi:

$$m^{ed} = m^{1 + k(p-1)(q-1)} = m \cdot (m^{(p-1)})^{k(q-1)}$$

Caso 1: Se $\gcd(m, p) = 1$, allora per il Piccolo Teorema di Fermat:

$$m^{(p-1)} \equiv 1 \pmod{p}$$

Quindi:

$$m^{ed} = m \cdot (m^{(p-1)})^{k(q-1)} \equiv m \cdot 1^{k(q-1)} \equiv m \pmod{p}$$

Caso 2: Se p divide m , allora $m \equiv 0 \pmod{p}$, quindi:

$$m^{ed} \equiv 0 \equiv m \pmod{p}$$

In entrambi i casi, $m^{ed} \equiv m \pmod{p}$. Analogamente, $m^{ed} \equiv m \pmod{q}$. Per il Teorema Cinese del Resto:

$$m^{ed} \equiv m \pmod{n}$$

Questo completa la dimostrazione.

Corollario importante: Se $\gcd(m, n) = 1$, allora possiamo lavorare nel gruppo $(\mathbb{Z}/n\mathbb{Z})^*$, e per il Teorema di Eulero $m^{\varphi(n)} \equiv 1 \pmod{n}$. Quindi:

$$m^{ed} = m^{1 + k\varphi(n)} = m \cdot (m^{\varphi(n)})^k \equiv m \cdot 1^k \equiv m \pmod{n}$$

Tuttavia, la dimostrazione precedente è più generale perché funziona anche quando $\gcd(m, n) \neq 1$, il che è importante per la completezza teorica (anche se nella pratica è estremamente improbabile che un messaggio casuale m abbia un fattore comune con n).

4.2 Schema Completo dell'Algoritmo RSA

Generazione delle chiavi:

1. **Selezione dei primi:** Genera due numeri primi distinti p e q , ciascuno di lunghezza circa $\ell/2$ bit, dove ℓ è la lunghezza desiderata del modulo (tipicamente $\ell = 2048$ o 3072).
 - Scegli p e q casualmente tra i primi di circa $\ell/2$ bit
 - Assicurati che $|p - q|$ sia sufficientemente grande (ad esempio, $|p - q| > 2^{\ell/2 - 100}$)
 - Questo previene attacchi di fattorizzazione che sfruttano fattori vicini
2. **Calcolo del modulo:** $n = p \cdot q$

Il modulo n è la parte pubblica e ha lunghezza ℓ bit.

3. **Calcolo di $\phi(n)$:** $\phi(n) = (p-1)(q-1) = n - p - q + 1$

Questo è il valore cruciale che deve rimanere segreto. La conoscenza di $\phi(n)$ è computazionalmente equivalente alla conoscenza della fattorizzazione di n .

4. **Scelta dell'esponente pubblico e :**

Scegli e tale che:

- $1 < e < \phi(n)$
- $\gcd(e, \phi(n)) = 1$

5. Valori comuni per e :

- $e = 3$: Molto efficiente per la cifratura, ma richiede padding adeguato
- $e = 17$: Buon compromesso
- $e = 65537 = 2^{16} + 1$: Valore raccomandato, efficiente (solo 2 bit a 1) e sicuro

6. La scelta di e piccolo rende la cifratura efficiente (pochi bit a 1 significa poche moltiplicazioni nell'esponenziazione veloce).

Calcolo dell'esponente privato d :

Calcola d tale che:

$$e \cdot d \equiv 1 \pmod{\phi(n)}$$

Questo si fa usando l'Algoritmo Esteso di Euclide per trovare l'inverso moltiplicativo di e modulo $\phi(n)$.

Algoritmo Esteso di Euclide:

Input: $e, \phi(n)$

Output: d tale che $e \cdot d \equiv 1 \pmod{\phi(n)}$

// Mantieni invariante: $e \cdot s_i + \phi(n) \cdot t_i = r_i$

$r_0 = e, \quad s_0 = 1, \quad t_0 = 0$

$r_1 = \phi(n), \quad s_1 = 0, \quad t_1 = 1$

$i = 1$

mentre $r_i \neq 0$:

$q_{i+1} = \lfloor r_{i-1} / r_i \rfloor$

$r_{i+1} = r_{i-1} - q_{i+1} \cdot r_i$

$s_{i+1} = s_{i-1} - q_{i+1} \cdot s_i$

$t_{i+1} = t_{i-1} - q_{i+1} \cdot t_i$

$i = i + 1$

```
// Ora  $r_{(i-1)} = \gcd(e, \varphi(n)) = 1$ 
//  $e \cdot s_{(i-1)} + \varphi(n) \cdot t_{(i-1)} = 1$ 
// Quindi  $e \cdot s_{(i-1)} \equiv 1 \pmod{\varphi(n)}$ 
```

se $s_{(i-1)} < 0$:

$d = s_{(i-1)} + \varphi(n)$

altrimenti:

$d = s_{(i-1)}$

restituisce d

7. **Complessità:** $O(\log^2 \varphi(n))$ operazioni bit, molto efficiente.

8. **Chiavi risultanti:**

- **Chiave pubblica:** (n, e)
- **Chiave privata:** (n, d) oppure (p, q, d) per ottimizzazioni

Cifratura:

Per cifrare un messaggio m con $0 \leq m < n$ usando la chiave pubblica (n, e):

$c = m^e \bmod n$

Calcolo efficiente di $m^e \bmod n$ (Esponenziazione binaria):

Input: m, e, n

Output: $c = m^e \bmod n$

Scrivi e in binario: $e = \sum_{i=0}^k e_i \cdot 2^i$ dove $e_i \in \{0,1\}$

$c = 1$

per i da k a 0:

$c = c^2 \bmod n$

se $e_i = 1$:

$c = c \cdot m \bmod n$

restituisce c

Analisi della complessità: Se e ha k bit, l'algoritmo richiede al massimo k moltiplicazioni modulari (da $c^2 \bmod n$) e al massimo k moltiplicazioni modulari (da $c \cdot m \bmod n$). Ogni moltiplicazione modulare di numeri di ℓ bit richiede $O(\ell^2)$ operazioni bit (con algoritmi classici) o $O(\ell \log \ell \log \log \ell)$ con algoritmi avanzati (Schönhage-Strassen).

Complessità totale: $O(k \cdot \ell^2) = O(\log e \cdot \log^2 n)$ operazioni bit.

Per e piccolo (come 65537), questo è molto efficiente. Ad esempio, $e = 65537$ ha solo 2 bit a 1, quindi richiede solo 16 moltiplicazioni modulo n per la parte di quadratura e 1 moltiplicazione per la parte moltiplicativa.

Decifratura:

Per decifrare un crittogramma c usando la chiave privata (n, d) :

$$m = c^d \bmod n$$

Il calcolo di $c^d \bmod n$ usa lo stesso algoritmo di esponenziazione binaria, ma poiché d è tipicamente grande (vicino a n), questo richiede circa $\log n$ moltiplicazioni modulari, quindi è più lento della cifratura quando e è piccolo.

Ottimizzazione della decifratura usando il Teorema Cinese del Resto:

Poiché conosciamo la fattorizzazione $n = pq$, possiamo accelerare la decifratura usando il TCR:

1. Calcola: $d_p = d \bmod (p-1)$ $d_q = d \bmod (q-1)$
2. Calcola: $m_p = c^{d_p} \bmod p$ $m_q = c^{d_q} \bmod q$
3. Ricostruisci m usando il TCR:

Metodo efficiente (usando il coefficiente cinese $q_{\text{inv}} = q^{-1} \bmod p$):

$$h = ((m_p - m_q) \cdot q_{\text{inv}}) \bmod p \quad m = m_q + h \cdot q$$

Analisi del guadagno: Le esponenziazioni modulari m_p e m_q lavorano con moduli di dimensione circa $n^{1/2}$ invece di n . Poiché la complessità dell'esponenziazione è $O(\log^3 n)$ (considerando il numero di moltiplicazioni e il costo di ciascuna), lavorare con moduli di dimensione $n^{1/2}$ riduce il costo di un fattore circa $(n^{1/2})^3 / (2 \cdot n^3) \approx 1/4$. Quindi la decifratura con il TCR è circa 4 volte più veloce della decifratura diretta.

4.3 Correttezza e Sicurezza di RSA

Teorema di correttezza:

Per ogni messaggio m con $0 \leq m < n$, la decifratura di $c = m^e \bmod n$ produce il messaggio originale m :

$$c^d \equiv (m^e)^d \equiv m^{ed} \equiv m \pmod{n}$$

Questo segue immediatamente dal Teorema fondamentale di RSA dimostrato precedentemente.

Assunzione di sicurezza RSA:

La sicurezza di RSA si basa su diverse assunzioni di difficoltà computazionale:

1. **Assunzione di fattorizzazione:** Dato $n = pq$, è computazionalmente intrattabile trovare p e q .
2. **Assunzione RSA (più forte):** Dato n , e , e $c = m^e \bmod n$, è computazionalmente intrattabile trovare m senza conoscere d .
3. **Assunzione della radice e-esima:** Dato n , e , e y , è computazionalmente intrattabile trovare x tale che $x^e \equiv y \pmod{n}$.

Relazioni tra le assunzioni:

- Se possiamo fattorizzare n , possiamo calcolare $\phi(n)$, quindi d , quindi possiamo decifrare. Quindi: Fattorizzazione $\Rightarrow$ Rottura di RSA.
- Non è noto se la rottura di RSA implichi la capacità di fattorizzare n . È una questione aperta importante se RSA sia equivalente alla fattorizzazione.
- Esistono "backdoor" teorici: se conosciamo $\phi(n)$ ma non la fattorizzazione, possiamo ancora decifrare. Ma calcolare $\phi(n)$ senza fattorizzare è considerato equivalente a fattorizzare.

Attacchi matematici contro RSA:

1. **Attacco con esponente piccolo e stesso messaggio:** Se lo stesso messaggio m è inviato a k destinatari con lo stesso esponente pubblico e (ma moduli diversi n_i), e se $k \geq e$, allora m può essere recuperato usando il Teorema Cinese del Resto.

Dettagli dell'attacco: Supponiamo che $e = 3$ e che m sia inviato a 3 destinatari con moduli n_1, n_2, n_3 . Allora:

$$c_1 \equiv m^3 \pmod{n_1} \quad c_2 \equiv m^3 \pmod{n_2} \quad c_3 \equiv m^3 \pmod{n_3}$$

Per il TCR, esiste un unico x modulo $N = n_1 n_2 n_3$ tale che:

$$x \equiv c_1 \pmod{n_1} \quad x \equiv c_2 \pmod{n_2} \quad x \equiv c_3 \pmod{n_3}$$

Questo x è esattamente m^3 (senza riduzione modulo N , perché $m < n_i$ per ogni i implica $m^3 < n_1 n_2 n_3$). Quindi possiamo calcolare m come la radice cubica intera di x .

Contromisura: Usare padding randomizzato (come OAEP) che rende ogni messaggio unico.

2. **Attacco con fattori comuni:** Se due utenti scelgono moduli n_1 e n_2 che condividono un fattore primo, allora $\gcd(n_1, n_2)$ rivela questo fattore.

Se p divide sia n_1 che n_2 , allora $p = \gcd(n_1, n_2)$, che può essere calcolato efficientemente con l'Algoritmo di Euclide in tempo $O(\log^2 n)$.

Probabilità teorica: Se i primi sono scelti veramente a caso, questa probabilità è trascurabile. Ma errori nell'implementazione dei generatori di numeri casuali hanno portato a vulnerabilità pratiche.

3. **Attacco di Wiener:** Se $d < n^{1/4} / 3$, allora d può essere trovato efficientemente usando l'algoritmo delle frazioni continue.

Idea dell'attacco: Da $ed \equiv 1 \pmod{\phi(n)}$, sappiamo che $ed = 1 + k\phi(n)$ per qualche k . Quindi:

$$e/\phi(n) \approx k/d$$

Poiché $\phi(n) = n - p - q + 1 \approx n$, abbiamo:

$$e/n \approx k/d$$

Se d è piccolo, k/d è una buona approssimazione razionale di e/n . L'algoritmo delle frazioni continue trova efficientemente le migliori approssimazioni razionali.

Dettagli matematici: Si può dimostrare che se $d < n^{1/4}/3$, allora k/d è un convergente dello sviluppo in frazioni continue di e/n . Calcolando i convergenti e testando ciascuno, possiamo trovare d .

Contromisura: Scegliere $d > n^{1/4}$, il che non è restrittivo in pratica.

4. **Attacco di Coppersmith:** Se conosciamo una frazione significativa dei bit di m o d , possiamo recuperare il valore completo usando metodi basati sui reticoli.
5. **Attacchi timing e side-channel:** Implementazioni ingenue possono rivelare informazione attraverso il tempo di esecuzione, consumo energetico, o emissioni elettromagnetiche.

4.4 Padding e RSA nella Pratica

Il RSA "testbook" (cioè $m^e \bmod n$ direttamente) non è sicuro nella pratica per diversi motivi:

1. **Determinismo:** Lo stesso messaggio produce sempre lo stesso crittogramma
2. **Malleabilità:** Dati $c_1 = m_1^e$ e $c_2 = m_2^e$, possiamo calcolare $c_3 = c_1 c_2 = (m_1 m_2)^e$
3. **Attacchi con messaggi scelti:** Varie vulnerabilità permettono attacchi

PKCS#1 v1.5 Padding (obsoleto ma ancora usato):

Prima della cifratura, il messaggio m è trasformato in:

$EM = 0x00 \parallel 0x02 \parallel PS \parallel 0x00 \parallel m$

dove:

- PS è una stringa di byte casuali non nulli, lunga almeno 8 byte
- La lunghezza totale di EM è uguale alla lunghezza di n in byte

RSA-OAEP (Optimal Asymmetric Encryption Padding) (standard moderno):

OAEP usa due funzioni hash G e H per trasformare il messaggio prima della cifratura:

Input: messaggio m, etichetta L (opzionale)

Parametri: lunghezza di n in bit: k, lunghezza hash: h

1. IHash = Hash(L)
2. PS = stringa di zeri di lunghezza $k - |m| - 2h - 2$
3. DB = IHash $\parallel$ PS $\parallel$ 0x01 $\parallel$ m
4. seed = stringa casuale di h bit
5. dbMask = MGF(seed, $k - h - 1$) // MGF = Mask Generation Function
6. maskedDB = DB $\oplus$ dbMask
7. seedMask = MGF(maskedDB, h)
8. maskedSeed = seed $\oplus$ seedMask
9. EM = 0x00 $\parallel$ maskedSeed $\parallel$ maskedDB
10. c = $EM^e \bmod n$

La Mask Generation Function (MGF) è tipicamente implementata come:

$MGF(\text{seed}, \text{len}) = \text{Hash}(\text{seed} \parallel 0) \parallel \text{Hash}(\text{seed} \parallel 1) \parallel \dots$ (troncato a len byte)

Proprietà di sicurezza di OAEP:

- **IND-CCA2 sicuro:** OAEP rende RSA sicuro contro attacchi a chosen-ciphertext adattivi, assumendo che RSA sia una permutazione one-way e che le funzioni hash siano modellate come oracoli casuali.
- **Randomizzazione:** Il seed casuale garantisce che lo stesso messaggio produca sempre crittogrammi diversi.
- **Integrità:** La struttura con IHash e il padding specifico permette di rilevare modifiche.

5. Il Protocollo Diffie-Hellman

Il protocollo di scambio chiavi Diffie-Hellman (1976) è stato il primo metodo pratico per stabilire una chiave segreta condivisa su un canale pubblico non sicuro.

5.1 Il Protocollo Classico DH

Setup del dominio pubblico:

1. Si sceglie un primo grande p (tipicamente 2048 o 3072 bit)
2. Si sceglie un generatore g del gruppo $(\mathbb{Z}/p\mathbb{Z})^*$ (o di un sottogruppo ciclico di ordine primo q che divide $p-1$)

I parametri (p, g) sono pubblici e possono essere condivisi da molti utenti.

Protocollo di scambio chiavi:

Alice e Bob vogliono stabilire una chiave segreta condivisa.

1. **Alice:**
 - Sceglie casualmente a con $1 \leq a < p-1$
 - Calcola $A = g^a \bmod p$
 - Invia A a Bob (su canale pubblico)
2. **Bob:**
 - Sceglie casualmente b con $1 \leq b < p-1$
 - Calcola $B = g^b \bmod p$
 - Invia B ad Alice (su canale pubblico)
3. **Alice calcola il segreto condiviso:**
 - $s = B^a \bmod p = (g^b)^a \bmod p = g^{(ab)} \bmod p$
4. **Bob calcola il segreto condiviso:**
 - $s = A^b \bmod p = (g^a)^b \bmod p = g^{(ab)} \bmod p$

Entrambi hanno calcolato lo stesso valore $s = g^{(ab)} \bmod p$, che può essere usato come chiave crittografica condivisa.

Correttezza matematica:

La correttezza si basa sulla proprietà commutativa dell'esponenziazione:

$$(g^a)^b = g^{(ab)} = g^{(ba)} = (g^b)^a$$

Dimostrazione formale:

Sia $G = \langle g \rangle$ un gruppo ciclico di ordine n . Per definizione di esponenziazione:

$$g^a = g \cdot g \cdot \dots \cdot g \text{ (a volte)}$$

Quindi:

$$\begin{aligned} (g^a)^b &= (g \cdot g \cdot \dots \cdot g)^b \text{ (a fattori)} \\ &= (g \cdot g \cdot \dots \cdot g) \cdot (g \cdot g \cdot \dots \cdot g) \cdot \dots \cdot (g \cdot g \cdot \dots \cdot g) \text{ (b gruppi di a)} \\ &= g \cdot g \cdot \dots \cdot g \text{ (ab volte)} \\ &= g^{(ab)} \end{aligned}$$

Per simmetria:

$$g^{(ab)} = g^{(ba)} = (g^b)^a$$

Estensioni del protocollo base

Diffie-Hellman con più di due parti:

Per n partecipanti $P_1, \dots, P_n$ che vogliono stabilire una chiave comune:

Protocollo in anello (Burmeister-Desmedt):

Round 1: Ogni P_i

- Sceglie r_i casuale
- Broadcast $z_i = g^{(r_i)}$

Round 2: Ogni P_i

- Riceve $z_{(i-1)}$ e $z_{(i+1)}$
- Broadcast $X_i = (z_{(i+1)} / z_{(i-1)})^{(r_i)}$

Calcolo chiave:

$$K = (z_1)^{(n \cdot r_n)} \cdot X_n^{(n-1)} \cdot X_{(n-1)}^{(n-2)} \cdot \dots \cdot X_2^1$$

Ogni partecipante può calcolare la stessa K usando i valori pubblici.

Protocollo ad albero:

Struttura gerarchica più efficiente per grandi n :

- Complessità comunicazione: $O(n)$ invece di $O(n^2)$
- Struttura ad albero binario
- Ogni nodo calcola chiave intermedia

Parametri di sicurezza raccomandati (2024)

Lunghezza di p (bit) | Lunghezza di q (bit) | Sicurezza equivalente

- 2048 | 224 | ~112 bit
- 3072 | 256 | ~128 bit
- 7680 | 384 | ~192 bit
- 15360 | 512 | ~256 bit

Raccomandazioni pratiche:

1. **Per nuovi sistemi:** Minimo 3072 bit (128 bit di sicurezza)
2. **Per sistemi legacy:** Migrazione da 2048 a 3072 bit entro 2030
3. **Per sicurezza a lungo termine:** 7680 bit o curve ellittiche

Attacco Small Subgroup (completamento)

Se g genera un sottogruppo piccolo invece dell'intero gruppo, informazioni possono trapelare.

Esempio di attacco:

Supponiamo $p = 2q + 1$ con q primo, e qualcuno usa erroneamente $g = 1$. Allora:

- $A = 1^a = 1$
- $B = 1^b = 1$
- Segreto condiviso $= 1^{(ab)} = 1$

Tutti i "segreti" sono uguali a 1!

Attacco più sottile:

Se g ha ordine piccolo d (es. $d = 2$, quindi $g^2 = 1$):

- Il segreto condiviso può assumere solo d valori
- Attaccante può enumerare tutti: solo $O(d)$ tentativi

Esempio numerico:

Sia $p = 23$, quindi $(\mathbb{Z}/23\mathbb{Z})^*$ ha ordine $22 = 2 \cdot 11$.

Se erroneamente $g = 22$ (che ha ordine 2):

$$22^2 = 484 \equiv 1 \pmod{23}$$

Allora per qualunque a, b :

- Se a pari: $g^a = 1$
- Se a dispari: $g^a = 22$
- Solo 2 possibili segreti condivisi!

Contromisura dettagliata:

Verificare che $g^q \equiv 1 \pmod{p}$ ma $g^{(q/r)} \not\equiv 1 \pmod{p}$ per ogni fattore primo r di q .

Per $p = 2q + 1$ (primo sicuro):

- Verificare $g^q \equiv 1 \pmod{p}$
- Verificare $g \not\equiv 1 \pmod{p}$
- Questo garantisce che g genera il sottogruppo di ordine q

Test di appartenenza al sottogruppo:

Quando si riceve $A = g^a$ da Alice, verificare:

$$A^q \equiv 1 \pmod{p}$$

Se fallisce, A non appartiene al sottogruppo corretto e potrebbe essere un attacco.

Parte IV: Crittografia su Curve Ellittiche e Protocolli Avanzati

$$(g^a)^b = g^{(ab)} = g^{(ba)} = (g^b)^a$$

Sicurezza di Diffie-Hellman:

Un attaccante passivo (eavesdropper) che osserva la comunicazione vede:

- I parametri pubblici p, g
- I valori scambiati $A = g^a \bmod p$ e $B = g^b \bmod p$

Per calcolare il segreto condiviso $s = g^{(ab)} \bmod p$, l'attaccante deve risolvere il problema Diffie-Hellman computazionale (CDH).

Alternativamente, se l'attaccante può risolvere il logaritmo discreto, può calcolare a da A (oppure b da B), e poi calcolare s .

Quindi la sicurezza di DH richiede:

1. Il problema del logaritmo discreto sia difficile in $(\mathbb{Z}/p\mathbb{Z})^*$
2. Il problema CDH sia difficile in $(\mathbb{Z}/p\mathbb{Z})^*$

Parametri sicuri per DH:

Per garantire la sicurezza, non basta scegliere p primo qualunque:

1. **Primo sicuro:** Si preferisce che p sia un "primo sicuro", cioè $p = 2q + 1$ dove q è anch'esso primo. In questo caso, $(\mathbb{Z}/p\mathbb{Z})^*$ ha ordine $2q$, quindi ha un sottogruppo ciclico di ordine primo q molto grande. Lavorare in questo sottogruppo evita l'attacco di Pohlig-Hellman.
2. **Dimensione del sottogruppo:** Se si usa un sottogruppo di ordine q che divide $p-1$, allora q deve essere grande (almeno 256 bit) per resistere agli attacchi generici sul DLP.
3. **Dimensione di p :** Per resistere agli algoritmi sub-esponenziali (index calculus), p deve essere almeno 2048 bit, preferibilmente 3072 bit o più.

Scelta del generatore:

Se $p = 2q + 1$ con q primo, per trovare un generatore del sottogruppo di ordine q :

1. Scegli h casualmente con $2 \leq h \leq p-2$
2. Calcola $g = h^2 \bmod p$
3. Se $g \neq 1$, allora g genera il sottogruppo di ordine q

Dimostrazione: L'ordine di g deve dividere $p-1 = 2q$. Poiché $g = h^2$ e $g \neq 1$, l'ordine di g non è 1 o 2. Quindi l'ordine di g è q o $2q$. Ma g è un quadrato, quindi $g^q = (h^2)^q = h^{2q} = h^{p-1} \equiv 1 \pmod{p}$ per il Piccolo Teorema di Fermat. Quindi l'ordine di g divide q . Combinando con il fatto che l'ordine divide $2q$ e non può essere 1 o 2, deve essere q .

5.2 Varianti di Diffie-Hellman**Diffie-Hellman Effimero (Ephemeral DH):**

In questa variante, sia Alice che Bob generano nuove chiavi private per ogni sessione. Questo fornisce forward secrecy: anche se le chiavi private vengono compromesse in futuro, le sessioni passate rimangono sicure.

Diffie-Hellman Statico-Effimero:

Una parte usa una chiave pubblica statica (certificata), mentre l'altra usa una chiave effimera. Questo permette autenticazione di una parte mentre mantiene alcune proprietà di forward secrecy.

Triple Diffie-Hellman (3-DH):

Combinazione di tre scambi DH per fornire autenticazione reciproca e forward secrecy. Usato in protocolli come Signal.

Siano A_s, B_s le chiavi pubbliche statiche di Alice e Bob, e A_e, B_e le chiavi effimere. Si calcolano:

$$k_1 = \text{DH}(A_s, B_e) \quad k_2 = \text{DH}(A_e, B_s) \quad k_3 = \text{DH}(A_e, B_e)$$

La chiave finale è $K = \text{KDF}(k_1 \parallel k_2 \parallel k_3)$ dove KDF è una funzione di derivazione chiave.

Proprietà di sicurezza:

- k_3 fornisce forward secrecy
- k_1 e k_2 forniscono autenticazione reciproca
- La combinazione protegge contro attacchi man-in-the-middle

5.3 Attacchi e Contromisure**Attacco Man-in-the-Middle (MITM):**

Il protocollo DH base è vulnerabile a MITM:

1. Mallory intercetta A da Alice
2. Mallory invia $M_1 = g^{a_1}$ ad Alice
3. Mallory intercetta B da Bob
4. Mallory invia $M_2 = g^{a_2}$ a Bob
5. Alice calcola $s_1 = M_1^a = g^{(am_1)}$
6. Bob calcola $s_2 = M_2^b = g^{(bm_2)}$
7. Mallory può decifrare e ricifrare tutti i messaggi

Contromisure:

1. **Autenticazione:** Firmare i valori scambiati con firme digitali
2. **Certificati:** Usare un'infrastruttura PKI per autenticare le chiavi pubbliche
3. **Password Authenticated Key Exchange (PAKE):** Protocolli che assumono una password condivisa

Attacco di Pohlig-Hellman sul DLP:

Se l'ordine del gruppo n ha solo fattori primi piccoli, il DLP può essere ridotto a istanze più piccole.

Supponiamo che $n = p_1^{e_1} \cdot p_2^{e_2} \cdot \dots \cdot p_k^{e_k}$ dove i p_i sono primi.

Per calcolare $\log_g(h)$ in un gruppo di ordine n :

1. Per ogni i , calcola $x_i = \log_g(h) \bmod p_i^{e_i}$ risolvendo un'istanza DLP in un gruppo di ordine $p_i^{e_i}$
2. Usa il Teorema Cinese del Resto per ricostruire $x = \log_g(h) \bmod n$

Complessità: $O(\sum_i (e_i(\log n + \sqrt{p_i})))$

Se tutti i p_i sono piccoli, questo è polinomiale!

Contromisura: Usare gruppi il cui ordine è primo o ha un fattore primo molto grande (almeno 256 bit).

Attacchi Small Subgroup:

Se g genera un sottogruppo piccolo invece dell'intero gruppo, informazioni possono trapelare.

Esempio: Supponiamo $p = 2q + 1$ con q primo, e qualcuno usa erroneamente $g = 1$. Allora tutti i "segreti condivisi" sono 1!

Più sottilmente, se g ha ordine piccolo d , allora il segreto condiviso può assumere solo d valori possibili, che possono essere enumerati.

Contromisura: Verificare sempre che $g^q \equiv 1 \pmod{p}$ ma $g^{(q/r)} \not\equiv 1 \pmod{p}$ per ogni fattore primo r di q .

6. Crittografia su Curve Ellittiche (ECC)

6.1 ECDH (Elliptic Curve Diffie-Hellman)

Setup del dominio:

1. Scegli una curva ellittica E su $\mathbb{F}_p$: $y^2 = x^3 + ax + b$
2. Scegli un punto base $G \in E(\mathbb{F}_p)$ di ordine primo n grande
3. I parametri del dominio (p, a, b, G, n) sono pubblici

Protocollo ECDH:

1. **Alice:**
 - Sceglie casualmente a con $1 \leq a < n$
 - Calcola $A = aG$ (moltiplicazione scalare sulla curva)
 - Invia A a Bob
2. **Bob:**
 - Sceglie casualmente b con $1 \leq b < n$
 - Calcola $B = bG$
 - Invia B ad Alice
3. **Segreto condiviso:**
 - Alice calcola $S = aB = a(bG) = (ab)G$
 - Bob calcola $S = bA = b(aG) = (ab)G$
 - La coordinata x del punto S è usata come chiave condivisa (dopo una funzione hash)

Calcolo della moltiplicazione scalare kG :

La moltiplicazione scalare è l'operazione fondamentale nell'ECC. L'algoritmo più comune è il double-and-add (analogo all'esponenziazione binaria):

Input: scalare k , punto G

Output: $P = kG$

Scrivi k in binario: $k = \sum_i k_i \cdot 2^i$

$P = O$ (punto neutro)

$Q = G$

per i da 0 a $\ell-1$: // ℓ = numero di bit di k

se $k_i = 1$:

$P = P + Q$

$Q = 2Q$ // raddoppio

restituisce P

Versione più efficiente (scansione da sinistra a destra):

$P = O$

per i da $\ell-1$ a 0 : // dal bit più significativo

$P = 2P$

se $k_i = 1$:

$P = P + G$

restituisce P

Complessità: $O(\log k)$ operazioni sulla curva (addizioni e raddoppi).

Formule esplicite per l'addizione su curve ellittiche:

Ricordiamo le formule per $E: y^2 = x^3 + ax + b$ su $\mathbb{F}_p$.

Addizione $P + Q$ con $P \neq Q$:

Siano $P = (x_1, y_1)$, $Q = (x_2, y_2)$ con $x_1 \neq x_2$.

$\lambda = (y_2 - y_1) \cdot (x_2 - x_1)^{-1} \bmod p$
 $x_3 = \lambda^2 - x_1 - x_2 \bmod p$
 $y_3 = \lambda(x_1 - x_3) - y_1 \bmod p$

Quindi $P + Q = (x_3, y_3)$.

Raddoppio $2P$:

Sia $P = (x_1, y_1)$ con $y_1 \neq 0$.

$\lambda = (3x_1^2 + a) \cdot (2y_1)^{-1} \bmod p$
 $x_3 = \lambda^2 - 2x_1 \bmod p$
 $y_3 = \lambda(x_1 - x_3) - y_1 \bmod p$

Quindi $2P = (x_3, y_3)$.

Costo computazionale:

- Un'addizione richiede: 1 inversione modulare, 2 moltiplicazioni, 1 elevamento al quadrato, poche addizioni/sottrazioni
- Un raddoppio richiede: 1 inversione modulare, 2 moltiplicazioni, 2 elevamenti al quadrato, poche addizioni/sottrazioni

L'inversione modulare è l'operazione più costosa. Usando l'algoritmo esteso di Euclide, costa $O(\log^2 p)$.

Ottimizzazione: Coordinate proiettive:

Per evitare inversioni durante il calcolo, si usano coordinate proiettive.

Un punto (x, y) in coordinate affini è rappresentato da (X, Y, Z) in coordinate proiettive dove:

$x = X/Z$, $y = Y/Z$

Il punto all'infinito O è rappresentato da $(0, 1, 0)$.

L'equazione della curva diventa:

$$Y^2Z = X^3 + aXZ^2 + bZ^3$$

Addizione in coordinate proiettive (formula di esempio):

Siano $P = (X_1, Y_1, Z_1)$, $Q = (X_2, Y_2, Z_2)$.

$$U_1 = X_1Z_2^2 \quad U_2 = X_2Z_1^2 \quad S_1 = Y_1Z_2^3 \quad S_2 = Y_2Z_1^3 \quad H = U_2 - U_1 \quad R = S_2 - S_1$$

$$X_3 = R^2 - H^3 - 2U_1H^2 \quad Y_3 = R(U_1H^2 - X_3) - S_1H^3 \quad Z_3 = Z_1Z_2H$$

Tutto modulo p .

Vantaggio: Nessuna inversione modulare! Solo moltiplicazioni modulari, che sono molto più veloci.

Raddoppio in coordinate proiettive Jacobiane:

Le coordinate Jacobiane (X, Y, Z) rappresentano $(X/Z^2, Y/Z^3)$.

Formula per il raddoppio (assumendo $a = -3$ per ottimizzazione):

$$S = 4XY^2 \quad M = 3X^2 + aZ^4 = 3(X + Z^2)(X - Z^2) \quad // \text{ quando } a = -3 \quad X_3 = M^2 - 2S \quad Y_3 = M(S - X_3) - 8Y^4 \\ Z_3 = 2YZ$$

Costo: 4 moltiplicazioni, 4 elevamenti al quadrato (molto più efficiente dell'inversione).

6.2 ECDSA (Elliptic Curve Digital Signature Algorithm)

ECDSA è l'analogo su curve ellittiche del DSA (Digital Signature Algorithm).

Setup:

- Parametri del dominio: (p, a, b, G, n, h) dove h è il cofattore
- Chiave privata: intero d con $1 \leq d < n$
- Chiave pubblica: punto $Q = dG$

Generazione della firma:

Per firmare un messaggio m :

1. Calcola $e = \text{Hash}(m)$ e sia z i primi $\log_2(n)$ bit di e
2. Scegli k casualmente con $1 \leq k < n$ (cruciale: k deve essere fresco e segreto per ogni firma!)
3. Calcola $P = kG = (x_P, y_P)$
4. Calcola $r = x_P \bmod n$. Se $r = 0$, riparti dal passo 2
5. Calcola $s = k^{-1}(z + rd) \bmod n$. Se $s = 0$, riparti dal passo 2
6. La firma è (r, s)

Verifica della firma:

Per verificare la firma (r, s) sul messaggio m con chiave pubblica Q :

1. Verifica che $r, s \in [1, n-1]$
2. Calcola $e = \text{Hash}(m)$ e sia z i primi $\log_2(n)$ bit di e
3. Calcola $w = s^{-1} \bmod n$
4. Calcola $u_1 = zw \bmod n$ e $u_2 = rw \bmod n$
5. Calcola $P = u_1G + u_2Q$
6. La firma è valida se e solo se $r \equiv x_P \bmod n$

Dimostrazione della correttezza:

$$P = u_1G + u_2Q = u_1G + u_2(dG) = (u_1 + u_2d)G = (zw + rwd)G = (zs^{-1} + rds^{-1})G = s^{-1}(z + rd)G$$

Dalla generazione della firma: $s = k^{-1}(z + rd) \bmod n$, quindi:

$$s^{-1} = k(z + rd)^{-1} \bmod n$$

Ma vogliamo dimostrare che $s^{-1}(z + rd) \equiv k \bmod n$:

$$s^{-1}(z + rd) = k(z + rd)^{-1}(z + rd) = k \bmod n$$

Quindi:

$$P = kG$$

che è esattamente il punto calcolato durante la generazione della firma. Quindi $x_P \equiv r \bmod n$, e la verifica ha successo.

Criteri di sicurezza critici:

1. **Randomness di k:** Se k è prevedibile o riutilizzato, la chiave privata può essere calcolata!

Attacco con k riutilizzato: Se due firme (r, s_1) e (r, s_2) usano lo stesso k (quindi stesso r), allora:

$$s_1 = k^{-1}(z_1 + rd) \bmod n \quad s_2 = k^{-1}(z_2 + rd) \bmod n$$

Sottraendo:

$$s_1 - s_2 = k^{-1}(z_1 - z_2) \bmod n$$

Quindi:

$$k = (z_1 - z_2)(s_1 - s_2)^{-1} \bmod n$$

Una volta ottenuto k , dalla prima equazione:

$$d = r^{-1}(ks_1 - z_1) \bmod n$$

La chiave privata è compromessa!

2. **Qualità dell'hash:** La funzione hash deve essere resistente alle collisioni e preimage.
3. **Protezione side-channel:** Implementazioni devono usare timing costante per evitare che informazioni su k o d trapelino attraverso il tempo di esecuzione.

ECDSA deterministico (RFC 6979):

Per evitare problemi con la generazione di k casuale, RFC 6979 definisce un metodo deterministico:

$$k = \text{HMAC-DRBG}(d, \text{Hash}(m))$$

dove HMAC-DRBG è un generatore di bit pseudo-casuali basato su HMAC. Questo garantisce che:

- k è deterministico dato d e m
- k appare casuale a chi non conosce d
- k è diverso per messaggi diversi

6.3 EdDSA e Curve25519

Curve di Edwards:

Le curve di Edwards sono una forma alternativa di curve ellittiche con proprietà computazionali migliori:

$$x^2 + y^2 = 1 + dx^2y^2$$

dove $d \in \mathbb{F}_p$ con $d \neq 0, 1$.

Legge di gruppo:

Per punti $P_1 = (x_1, y_1)$ e $P_2 = (x_2, y_2)$:

$$x_3 = (x_1y_2 + y_1x_2) / (1 + dx_1x_2y_1y_2) \quad y_3 = (y_1y_2 - x_1x_2) / (1 - dx_1x_2y_1y_2)$$

Vantaggi:

1. **Formula di addizione unificata:** La stessa formula funziona per addizione e raddoppio
2. **Completezza:** La formula funziona per tutti i punti (nessun caso speciale)
3. **Efficienza:** Le operazioni sono più veloci che su curve di Weierstrass

Curve25519:

Curve25519 è una curva di Montgomery (una variante) definita su $\mathbb{F}_p$ dove $p = 2^{255} - 19$:

$$y^2 = x^3 + 486662x^2 + x$$

Proprietà di Curve25519:

- $p = 2^{255} - 19$: forma speciale che permette riduzioni modulari molto efficienti
- Ordine del gruppo: $n = 8 \cdot \ell$ dove ℓ è un primo di 252 bit
- Sicurezza stimata: 128 bit
- Progettata per essere resistente agli attacchi timing

EdDSA su Ed25519:

Ed25519 è la variante Edwards equivalente di Curve25519:

$$-x^2 + y^2 = 1 - (121665/121666)x^2y^2$$

Schema di firma EdDSA:

1. **Chiave privata:** 256 bit casuali s
2. **Derivazione chiave:**
 - Calcola $h = \text{SHA-512}(s)$
 - I primi 256 bit di h definiscono lo scalare a (con clamping)
 - Gli ultimi 256 bit definiscono il prefix per la generazione del nonce
3. **Chiave pubblica:** $A = aB$ dove B è il punto base

Generazione firma:

1. $r = \text{SHA-512}(h[32:64] \parallel m) \bmod \ell$ (nonce deterministico)
2. $R = rB$
3. $k = \text{SHA-512}(R \parallel A \parallel m) \bmod \ell$
4. $S = (r + ka) \bmod \ell$
5. Firma: (R, S)

Verifica:

Verifica che:

$$SB = R + kA$$

dove $k = \text{SHA-512}(R \parallel A \parallel m) \bmod \ell$.

Vantaggi di EdDSA:

1. **Nonce deterministico:** Elimina vulnerabilità da cattiva generazione casuale
2. **Piccole firme:** 512 bit totali (64 byte)
3. **Prestazioni:** Molto veloce, specialmente la verifica
4. **Resistenza side-channel:** Progettato con attenzione per evitare timing attacks
5. **Completezza:** Nessun caso speciale nell'implementazione

7. Crittografia Post-Quantistica

7.1 La Minaccia Quantistica

Algoritmo di Shor:

Nel 1994, Peter Shor ha dimostrato che un computer quantistico può:

1. Fattorizzare interi in tempo polinomiale: $O((\log n)^3)$
2. Risolvere il logaritmo discreto in tempo polinomiale

Questo rompe completamente:

- RSA
- Diffie-Hellman
- DSA
- ECDH
- ECDSA

Algoritmo di Grover:

L'algoritmo di Grover fornisce un'accelerazione quadratica per la ricerca non strutturata. Per la crittografia simmetrica:

- Chiavi a 128 bit forniscono ~64 bit di sicurezza quantistica
- Chiavi a 256 bit forniscono ~128 bit di sicurezza quantistica

Quindi: La crittografia simmetrica è meno vulnerabile, ma le lunghezze delle chiavi devono essere raddoppiate.

7.2 Crittografia Basata su Reticoli (Lattices)

I reticoli sono una delle principali candidati per la crittografia post-quantistica.

Definizione di reticolo:

Un reticolo in $\mathbb{R}^n$ è un sottogruppo discreto generato da una base:

$$L(B) = \{Bx : x \in \mathbb{Z}^m\}$$

dove B è una matrice $n \times m$ di rango m .

Problema SVP (Shortest Vector Problem):

Dato un reticolo L , trovare un vettore non nullo $v \in L$ di norma minima.

Formalmente: trovare $v \in L \setminus \{0\}$ tale che $\|v\| = \lambda_1(L)$ dove:

$$\lambda_1(L) = \min\{\|v\| : v \in L, v \neq 0\}$$

Problema CVP (Closest Vector Problem):

Dato un reticolo L e un punto target $t \in \mathbb{R}^n$, trovare $v \in L$ che minimizza $\|v - t\|$.

Difficoltà computazionale:

Entrambi SVP e CVP sono NP-hard nel caso peggiore. Anche le versioni approssimate (trovare un vettore entro un fattore γ dall'ottimo) sono difficili per γ polinomiale.

Crucialmente, i migliori algoritmi quantistici conosciuti per questi problemi non offrono miglioramenti esponenziali rispetto agli algoritmi classici.

Learning With Errors (LWE):

LWE è un problema computazionale introdotto da Regev nel 2005, basato su reticoli.

Setup: Siano n, q, m parametri interi, e sia χ una distribuzione di errore su $\mathbb{Z}_q$ (tipicamente Gaussiana discreta).

Problema decisionale LWE: Distinguere tra:

1. Coppie (a_i, b_i) dove $a_i \in \mathbb{Z}_q^n$ è uniforme e $b_i = \langle a_i, s \rangle + e_i \bmod q$ per un $s \in \mathbb{Z}_q^n$ segreto fisso e $e_i \leftarrow \chi$
2. Coppie (a_i, b_i) completamente uniformi in $\mathbb{Z}_q^n \times \mathbb{Z}_q$

Problema di ricerca LWE: Date m coppie LWE (a_i, b_i) , trovare s .

Riduzione a reticoli:

Regev ha dimostrato che risolvere LWE (nella versione peggiore-media) è difficile quanto approssimare problemi sui reticoli nel caso peggiore (come GapSVP e SIVP).

Questa riduzione dal caso peggiore al caso medio è notevole e fornisce una forte garanzia di sicurezza.

Ring-LWE:

Per efficienza, si usa spesso la variante Ring-LWE, dove gli elementi sono polinomi in un anello quoziente:

$$R_q = \mathbb{Z}_q[x]/(x^n + 1)$$

dove n è tipicamente una potenza di 2.

Problema Ring-LWE: Simile a LWE ma con $s, a_i, e_i \in R_q$.

Vantaggio: Le operazioni su R_q possono essere implementate efficientemente usando la Number Theoretic Transform (NTT), riducendo la complessità da $O(n^2)$ a $O(n \log n)$.

7.3 Schema di Cifratura Basato su LWE**Schema semplificato:**

Generazione chiavi:

- Parametri pubblici: n, q, m , distribuzione χ
- Genera matrice $A \in \mathbb{Z}_q^{(n \times m)}$ uniforme
- Scegli $s \in \mathbb{Z}_q^n$ (chiave segreta)
- Scegli $e \in \mathbb{Z}_q^m$ da χ^m
- Calcola $b = As + e \bmod q$
- Chiave pubblica: (A, b)
- Chiave privata: s

Cifratura di $m \in \{0,1\}$:

- Scegli $r \in \{0,1\}^m$ casuale
- $c_1 = A^T r \bmod q$
- $c_2 = b^T r + m \cdot \lfloor q/2 \rfloor \bmod q$
- Crittogramma: (c_1, c_2)

Decifratura:

- Calcola $v = c_2 - s^T c_1 \bmod q$
- Se v è più vicino a 0 che a $\lfloor q/2 \rfloor$, output 0
- Altrimenti, output 1

Correttezza:

$$v = c_2 - s^T c_1 = b^T r + m \cdot \lfloor q/2 \rfloor - s^T (A^T r) = (As + e)^T r + m \cdot \lfloor q/2 \rfloor - s^T A^T r = s^T A^T r + e^T r + m \cdot \lfloor q/2 \rfloor - s^T A^T r = e^T r + m \cdot \lfloor q/2 \rfloor$$

Se $\|e^T r\| < \lfloor q/4 \rfloor$ (che accade con alta probabilità se l'errore è piccolo), allora:

- Quando $m = 0$: $v = e^T r \approx 0$
- Quando $m = 1$: $v = e^T r + \lfloor q/2 \rfloor \approx \lfloor q/2 \rfloor$

Quindi la decifratura recupera correttamente m .

Sicurezza:

La sicurezza si basa sull'impossibilità di distinguere $(A, As + e)$ da (A, u) dove u è uniforme, che è esattamente l'assunzione decisionale LWE.

7.4 NIST Post-Quantum Standardization

Nel 2016, il NIST ha iniziato un processo per standardizzare algoritmi crittografici post-quantistici. Nel 2024, i primi standard sono stati pubblicati:

CRYSTALS-Kyber (Cifratura a chiave pubblica):

- Basato su Module-LWE (generalizzazione di Ring-LWE)
- Livelli di sicurezza: Kyber-512 (128 bit), Kyber-768 (192 bit), Kyber-1024 (256 bit)
- Prestazioni eccellenti: cifratura e decifratura molto veloci
- Dimensioni chiavi moderate: chiave pubblica ~800-1568 byte, crittogramma ~768-1568 byte

CRYSTALS-Dilithium (Firme digitali):

- Basato su Module-LWE
- Livelli di sicurezza: Dilithium2 (128 bit), Dilithium3 (192 bit), Dilithium5 (256 bit)
- Firme di dimensione 2420-4595 byte
- Verifica molto veloce

FALCON (Firme digitali):

- Basato su NTRU lattices
- Firme più compatte di Dilithium (~666-1280 byte)
- Generazione chiavi più lenta
- Richiede aritmetica in virgola mobile ad alta precisione

SPHINCS+ (Firme digitali):

- Basato su funzioni hash (non reticoli)
- Completamente stateless
- Firme molto grandi (~7856-49856 byte)
- Molto conservativo: sicurezza basata solo su hash

8. Primitive Crittografiche Avanzate**8.1 Funzioni Hash Crittografiche**

Una funzione hash crittografica $H: \{0,1\}^* \rightarrow \{0,1\}^n$ mappa stringhe di lunghezza arbitraria in digest di lunghezza fissa n .

Proprietà richieste:

1. **Resistenza alla preimmagine (one-wayness):** Dato h , è computazionalmente intrattabile trovare m tale che $H(m) = h$.

Complessità attesa: $O(2^n)$ valutazioni di H .

Formalizzazione: Per ogni algoritmo PPT (tempo polinomiale probabilistico) A :

$$\Pr[H(A(h)) = h : h \leftarrow \{0,1\}^n] < \epsilon(n)$$

1. dove $\epsilon(n)$ è trascurabile.

2. **Resistenza alla seconda preimmagine:** Dato m_1 , è computazionalmente intrattabile trovare $m_2 \neq m_1$ tale che $H(m_1) = H(m_2)$.

Complessità attesa: $O(2^n)$ valutazioni.

Formalizzazione: Per ogni algoritmo PPT A :

$$\Pr[m_2 \leftarrow A(m_1) : m_2 \neq m_1 \wedge H(m_1) = H(m_2)] < \epsilon(n)$$

3. **Resistenza alle collisioni:** È computazionalmente intrattabile trovare qualunque coppia $m_1 \neq m_2$ tale che $H(m_1) = H(m_2)$.

Complessità attesa: $O(2^{n/2})$ per il paradosso del compleanno.

Formalizzazione: Per ogni algoritmo PPT A :

$$\Pr[(m_1, m_2) \leftarrow A() : m_1 \neq m_2 \wedge H(m_1) = H(m_2)] < \epsilon(n)$$

Relazioni tra le proprietà:

Resistenza collisioni $\Rightarrow$ Resistenza seconda preimmagine $\Rightarrow$ Resistenza preimmagine

Le implicazioni inverse non valgono necessariamente.

Dimostrazione: Collisioni $\Rightarrow$ Seconda preimmagine:

Se esiste un algoritmo A che rompe la resistenza alla seconda preimmagine, costruiamo un algoritmo B che trova collisioni:

1. Scegli m_1 casuale
2. Calcola $m_2 = A(m_1)$
3. Output (m_1, m_2)

Se A ha successo, allora B trova una collisione.

Parte V: Funzioni Hash, Codici MAC e Protocolli Avanzati

8. Primitive Crittografiche Avanzate (continuazione)

8.1 Funzioni Hash Crittografiche

Una funzione hash crittografica $H: \{0,1\}^* \rightarrow \{0,1\}^n$ mappa stringhe di lunghezza arbitraria in digest di lunghezza fissa n .

Proprietà richieste:

1. **Resistenza alla preimmagine (one-wayness):** Dato h , è computazionalmente intrattabile trovare m tale che $H(m) = h$. Complessità attesa: $O(2^n)$ valutazioni di H .
2. **Resistenza alla seconda preimmagine:** Dato m_1 , è computazionalmente intrattabile trovare $m_2 \neq m_1$ tale che $H(m_1) = H(m_2)$. Complessità attesa: $O(2^n)$ valutazioni.

3. **Resistenza alle collisioni:** È computazionalmente intrattabile trovare qualunque coppia $m_1 \neq m_2$ tale che $H(m_1) = H(m_2)$. Complessità attesa: $O(2^{n/2})$ per il paradosso del compleanno.

Il paradosso del compleanno:

Se scegliamo k elementi casuali da uno spazio di dimensione N , la probabilità di trovare una collisione è approssimativamente:

$$P(\text{collisione}) \approx 1 - e^{-(k^2/(2N))}$$

Per $P \approx 1/2$, abbiamo $k \approx \sqrt{(2N \cdot \ln 2)} \approx 1.17\sqrt{N}$.

Per un hash a n bit, $N = 2^n$, quindi $k \approx 1.17 \cdot 2^{n/2}$.

Implicazioni: Un hash da 256 bit fornisce solo 128 bit di sicurezza contro le collisioni, ma 256 bit contro attacchi di preimmagine.

Costruzione di Merkle-Damgård:

Molte funzioni hash classiche (MD5, SHA-1, SHA-2) usano questa costruzione:

1. **Funzione di compressione:** $f: \{0,1\}^n \times \{0,1\}^b \rightarrow \{0,1\}^n$ che mappa uno stato di n bit e un blocco di b bit in un nuovo stato di n bit.
2. **Padding:** Il messaggio M è paddato a una lunghezza multipla di b bit. Il padding include la lunghezza del messaggio originale (Merkle-Damgård strengthening).
3. **Iterazione:**
 - Inizializza $H_0 = IV$ (valore di inizializzazione fisso)
 - Per $i = 1$ a ℓ (numero di blocchi): $H_i = f(H_{i-1}, M_i)$
 - Output H_ℓ

Teorema di Merkle-Damgård: Se f è resistente alle collisioni, allora H è resistente alle collisioni.

Dimostrazione: Supponiamo che troviamo una collisione in H , cioè due messaggi M e M' con $M \neq M'$ tali che $H(M) = H(M')$. Siano $M = M_1 || \dots || M_\ell$ e $M' = M'_1 || \dots || M'_\ell$ dopo il padding.

Tracciamo gli stati intermedi:

- $M: H_0 \rightarrow H_1 \rightarrow \dots \rightarrow H_\ell$
- $M': H'_0 \rightarrow H'_1 \rightarrow \dots \rightarrow H'_\ell$

Poiché $H(M) = H(M')$, abbiamo $H_\ell = H'_\ell$.

Caso 1: Se $\ell \neq \ell'$, allora il padding include lunghezze diverse. Lavorando all'indietro dall'ultima collisione, troviamo necessariamente un indice i dove gli stati differiscono ma producono lo stesso output, fornendo una collisione per f .

Caso 2: Se $\ell = \ell'$ e $M_\ell \neq M'_\ell$, allora $f(H_{i-1}, M_\ell) = f(H'_{i-1}, M'_\ell)$. Se $H(\ell-1) = H'(\ell-1)$, abbiamo immediatamente una collisione per f . Altrimenti, continuiamo all'indietro fino a trovare il primo punto di divergenza.

Caso 3: Se $M_\ell = M'_\ell$, allora $H(\ell-1) = H'(\ell-1)$, e ripetiamo l'argomento. Poiché $M \neq M'$, eventualmente troviamo un blocco diverso, riducendoci ai casi precedenti.

In ogni caso, una collisione in H implica una collisione in f .

Vulnerabilità della costruzione Merkle-Damgård:

1. **Attacchi di estensione della lunghezza:** Dato $H(M)$, possiamo calcolare $H(M||\text{padding}||X)$ senza conoscere M . Questo è problematico per alcuni usi delle hash come MAC.
2. **Multicollisioni:** Se f ha una collisione, possiamo costruire 2^k messaggi diversi che collidono, scegliendo tra le due opzioni in k punti. Questo ha complessità $O(k \cdot 2^{n/2})$ invece di $O(2^{kn/(k+1)})$ per hash ideali.

SHA-256 (Secure Hash Algorithm 256-bit):

SHA-256 è parte della famiglia SHA-2, standardizzata da NIST nel 2001.

Parametri:

- Dimensione output: 256 bit
- Dimensione blocco: 512 bit
- Dimensione stato interno: 256 bit (otto word da 32 bit)
- Numero di round: 64

Funzione di compressione di SHA-256:

Lo stato è rappresentato da otto registri a 32 bit: a, b, c, d, e, f, g, h .

Per ogni round i (da 0 a 63):

$$T_1 = h + \Sigma_1(e) + \text{Ch}(e, f, g) + K_i + W_i$$

$$T_2 = \Sigma_0(a) + \text{Maj}(a, b, c)$$

$$h = g$$

$$g = f$$

$$f = e$$

$$e = d + T_1$$

$$d = c$$

$$c = b$$

$$b = a$$

$$a = T_1 + T_2$$

Dove:

- $\text{Ch}(e,f,g) = (e \wedge f) \oplus (\neg e \wedge g)$: "choose" - se e allora f, altrimenti g
- $\text{Maj}(a,b,c) = (a \wedge b) \oplus (a \wedge c) \oplus (b \wedge c)$: "majority" - valore che appare in almeno 2 dei 3 input
- $\Sigma_0(x) = \text{ROTR}^2(x) \oplus \text{ROTR}^{13}(x) \oplus \text{ROTR}^{22}(x)$: rotazioni e XOR per diffusione
- $\Sigma_1(x) = \text{ROTR}^6(x) \oplus \text{ROTR}^{11}(x) \oplus \text{ROTR}^{25}(x)$: rotazioni diverse per e
- K_i : costanti derivate dalle radici cubiche dei primi 64 numeri primi
- W_i : schedule dei messaggi

Message Schedule: I primi 16 valori $W_0 \dots W_{15}$ sono i 16 word da 32 bit del blocco di input.

Gli altri sono calcolati come:

$$W_i = \sigma_1(W_{i-2}) + W_{i-7} + \sigma_0(W_{i-15}) + W_{i-16}$$

dove:

- $\sigma_0(x) = \text{ROTR}^7(x) \oplus \text{ROTR}^{18}(x) \oplus \text{SHR}^3(x)$
- $\sigma_1(x) = \text{ROTR}^{17}(x) \oplus \text{ROTR}^{19}(x) \oplus \text{SHR}^{10}(x)$

Razionale del design:

1. Le funzioni Ch e Maj introducono non-linearità
2. Le rotazioni e shift forniscono diffusione bit-per-bit
3. Il message schedule propaga le modifiche attraverso tutti i round
4. Le costanti K prevengono simmetrie
5. L'uso di operazioni modulari a 32 bit è efficiente su hardware moderno

Sicurezza di SHA-256:

- Nessuna collisione trovata (migliori attacchi raggiungono 31 dei 64 round)
- Considerato sicuro contro tutti gli attacchi pratici noti
- Resistenza stimata: 256 bit contro preimmagine, 128 bit contro collisioni
- Sicurezza quantistica: ~85 bit contro l'algoritmo di Grover (ottimizzato)

SHA-3 (Keccak):

SHA-3, selezionato tramite competizione nel 2012 e standardizzato nel 2015, usa una costruzione completamente diversa chiamata "sponge construction".

Sponge Construction:

1. **Stato:** Un array di b bit, diviso in due parti:
 - r bit: "rate" (capacità di throughput)
 - c bit: "capacity" (sicurezza)

- $b = r + c$ (tipicamente $b = 1600$ per SHA-3)

2. Fase di assorbimento:

- Inizializza stato $S = 0^b$
- Per ogni blocco di input P_i di r bit:
 - $S = f(S \oplus (P_i \parallel 0^c))$
- f è una permutazione (la permutazione Keccak-f)

3. Fase di spremitura (squeezing):

- Output i primi r bit di S
- Se servono più bit: $S = f(S)$, output altri r bit
- Ripeti fino a ottenere la lunghezza desiderata

Permutazione Keccak-f[1600]:

Lo stato è visto come un array 3D: $S[5][5][64]$ di bit (totale 1600 bit).

Ogni round consiste di 5 step:

θ (theta): Mixing lineare tra colonne

$$\begin{aligned} C[x,z] &= \oplus_{y=0 \text{ a } 4} S[x,y,z] \\ D[x,z] &= C[x-1,z] \oplus C[x+1,z-1] \\ S[x,y,z] &= S[x,y,z] \oplus D[x,z] \end{aligned}$$

1.

ρ (rho): Rotazione di ciascuna lane di un offset specifico

$$S[x,y,*] = \text{ROT}(S[x,y,*], r[x,y])$$

2.

π (pi): Permutazione delle lane

$$B[y,2x+3y,z] = S[x,y,z]$$

3.

χ (chi): Unico step non-lineare

$$S[x,y,z] = B[x,y,z] \oplus ((\neg B[x+1,y,z]) \wedge B[x+2,y,z])$$

4.

ι (iota): Aggiunta di una costante di round

$$S[0,0,*] = S[0,0,*] \oplus \text{RC}[\text{round}]$$

5.

Numero di round: 24 per Keccak-f[1600]

Istanze di SHA-3:

- **SHA3-224:** $r = 1152$, $c = 448$, output = 224 bit
- **SHA3-256:** $r = 1088$, $c = 512$, output = 256 bit
- **SHA3-384:** $r = 832$, $c = 768$, output = 384 bit
- **SHA3-512:** $r = 576$, $c = 1024$, output = 512 bit

Sicurezza: La sicurezza è determinata dalla capacity c . Per resistenza alle collisioni: $2^{(c/2)}$.
Per resistenza alla preimmagine: 2^c .

Vantaggi di SHA-3:

1. Diversità algoritmica rispetto a SHA-2
2. Nessuna vulnerabilità agli attacchi di estensione della lunghezza
3. Può produrre output di lunghezza arbitraria (modalità SHAKE)
4. Struttura semplice e elegante
5. Buone proprietà di sicurezza dimostrabili

8.2 Message Authentication Codes (MAC)

Un MAC è una primitiva crittografica che fornisce autenticazione e integrità dei messaggi.

Definizione formale:

Un MAC è una tripla di algoritmi (Gen, Tag, Verify):

- **Gen:** Genera una chiave segreta k
- **Tag:** Dato k e messaggio m , produce un tag $\tau = \text{Tag}_k(m)$
- **Verify:** Dato k , m , τ , verifica se τ è un tag valido per m

Proprietà di sicurezza (EUF-CMA):

Un MAC è esistenzialmente inforgiabile sotto attacchi a messaggio scelto (EUF-CMA) se un avversario che può ottenere tag per messaggi di sua scelta non può produrre un tag valido per un nuovo messaggio con probabilità non trascurabile.

HMAC (Hash-based MAC):

HMAC è costruito usando una funzione hash crittografica H :

$$\text{HMAC}_K(m) = H((K \oplus \text{opad}) \parallel H((K \oplus \text{ipad}) \parallel m))$$

dove:

- **opad** = 0x5c ripetuto (outer padding)
- **ipad** = 0x36 ripetuto (inner padding)
- $\parallel$ denota concatenazione
- K è la chiave, eventualmente hashata e paddato alla lunghezza del blocco di H

Perché questa costruzione?

1. $H(K || m)$ da solo è vulnerabile: attacchi di estensione della lunghezza permettono di calcolare $H(K || m || x)$ dato $H(K || m)$.
2. $H(m || K)$ da solo è vulnerabile: collisioni interne possono essere sfruttate.
3. La **doppia hash con chiavi diverse** (derivate da K tramite XOR con costanti) previene questi attacchi.

Analisi di sicurezza:

Assumendo che H sia modellata come funzione pseudo-casuale (PRF), HMAC è EUF-CMA sicuro. La dimostrazione si basa su:

1. Se H è una PRF, allora $H(K || \cdot)$ è una PRF
2. La costruzione interna $H((K \oplus \text{ipad}) || m)$ produce un output pseudo-casuale
3. La costruzione esterna applica H nuovamente con una chiave derivata diversa
4. Un attaccante dovrebbe invertire almeno una delle due applicazioni di H

Teorema (Bellare, Canetti, Krawczyk): Se H è una famiglia di funzioni hash resistente alle collisioni e la funzione di compressione di H è una PRF, allora HMAC è un PRF sicuro.

CMAC (Cipher-based MAC):

CMAC usa un cifrario a blocchi (come AES) invece di una funzione hash.

Costruzione:

1. Dividi il messaggio in blocchi di n bit: $M = M_1 || M_2 || \dots || M_t$
2. Inizializza $C_0 = 0^n$
3. Per $i = 1$ a $t-1$: $C_i = E_K(C_{i-1} \oplus M_i)$
4. Se M_t è completo: $C_t = E_K(C_{t-1} \oplus M_t \oplus K_1)$ Altrimenti: $C_t = E_K(C_{t-1} \oplus (M_t || 10^*) \oplus K_2)$
5. Output C_t (o i primi τ bit per tag più corto)

Dove K_1 e K_2 sono derivate da K usando moltiplicazioni in $GF(2^n)$.

Perché K_1 e K_2 ?

Queste chiavi derivate prevengono attacchi di estensione. Senza di esse, un attaccante potrebbe:

- Dato $MAC(M)$, calcolare $MAC(M || M')$ per certi M'

Sicurezza di CMAC:

Se E è una PRP (Pseudo-Random Permutation) sicura, allora CMAC è EUF-CMA sicuro. La dimostrazione usa la tecnica del game-hopping:

1. Sostituisci E con una permutazione casuale π
2. Dimostra che l'avversario non può distinguere (bound sulla query)
3. Analizza la probabilità di forgery contro il MAC con permutazione casuale
4. Il bound risultante è $O(q^2/2^n)$ dove q è il numero di query

Poly1305:

Poly1305 è un MAC estremamente veloce progettato da Daniel Bernstein.

Costruzione matematica:

Sia r una chiave di 128 bit "clamped" (certi bit fissati a 0 per evitare attacchi). Sia s una chiave ausiliaria di 128 bit.

Il messaggio M è diviso in blocchi da 16 byte: $M_1, M_2, \dots, M_\ell$.

Ogni blocco M_i è interpretato come un intero little-endian e gli si aggiunge 2^{128} :

$$c_i = M_i + 2^{128} \text{ (tranne possibilmente l'ultimo blocco)}$$

Il MAC è calcolato come:

$$\text{Tag} = ((c_\ell \cdot r^\ell + c_{(\ell-1)} \cdot r^{(\ell-1)} + \dots + c_1 \cdot r) \bmod (2^{130}-5)) + s$$

Perché $2^{130}-5$?

È un numero primo vicino a una potenza di 2, permettendo riduzioni modulari efficienti.

Proprietà di sicurezza:

Poly1305 è un one-time MAC basato sulla valutazione polinomiale. La sicurezza si basa sul fatto che un polinomio di grado d ha al massimo d radici, quindi:

$$P(\text{forgery con } q \text{ query}) \leq d/p \approx q/(2^{130}-5) \approx q \cdot 2^{(-130)}$$

Velocità: Poly1305 è estremamente veloce su CPU moderne, raggiungendo ~5 cicli/byte.

8.3 Authenticated Encryption

La cifratura autenticata combina confidenzialità e autenticazione in un unico schema.

Modalità di composizione:

1. **Encrypt-and-MAC:** $E_{K_1}(M) \parallel \text{MAC}_{K_2}(M)$
 - Problematico: il MAC potrebbe rivelare informazioni su M
2. **MAC-then-Encrypt:** $E_{K_1}(M \parallel \text{MAC}_{K_2}(M))$
 - Vulnerabile a padding oracle attacks in certe implementazioni
3. **Encrypt-then-MAC:** $E_{K_1}(M) \parallel \text{MAC}_{K_2}(E_{K_1}(M))$

- Più sicuro: il MAC non vede il plaintext, e la verifica avviene prima della decifratura

Teorema (Bellare-Namprempre): Se E è IND-CPA sicuro e MAC è EUF-CMA sicuro, allora Encrypt-then-MAC è IND-CCA2 sicuro.

GCM (Galois/Counter Mode):

GCM è una modalità di authenticated encryption standardizzata e ampiamente usata.

Costruzione:

Cifratura (CTR mode):

Per $i = 1$ a n :

$$Y_i = E_K(\text{Counter}_i)$$

$$C_i = P_i \oplus Y_i$$

1.

Autenticazione (GHASH):

$$\text{GHASH}_H(A, C) = (A_1 \cdot H^m + A_2 \cdot H^{(m-1)} + \dots + C_1 \cdot H^n + \dots + C_n \cdot H + \text{len}(A, C)) \bmod p(x)$$

2. Dove:

- $H = E_K(0^{128})$ (chiave di autenticazione derivata)
- A è authenticated associated data (AAD)
- Le operazioni sono in $GF(2^{128})$ con polinomio $p(x) = x^{128} + x^7 + x^2 + x + 1$
- $\text{len}(A, C)$ codifica le lunghezze di A e C

Tag:

$$T = \text{MSB}_T(\text{GHASH}_H(A, C) \oplus E_K(\text{Counter}_0))$$

3.

Matematica di $GF(2^{128})$:

Gli elementi sono polinomi di grado ≤ 127 con coefficienti in $GF(2) = \{0, 1\}$.

Addizione: XOR bit a bit (equivalente a addizione di polinomi mod 2)

Moltiplicazione: Moltiplicazione di polinomi seguita da riduzione modulo $p(x)$

Esempio con rappresentazione bit:

$$a(x) = x^3 + x + 1 \rightarrow 1011 \text{ (binary)}$$

$$b(x) = x^2 + 1 \rightarrow 0101 \text{ (binary)}$$

$$\begin{aligned}
 a \cdot b &= (x^3 + x + 1)(x^2 + 1) \\
 &= x^5 + x^3 + x^3 + x + x^2 + 1 \\
 &= x^5 + x^2 + x + 1 \pmod{2}
 \end{aligned}$$

Implementazione efficiente:

La moltiplicazione in $GF(2^{128})$ può essere implementata usando:

1. **Metodo shift-and-xor:** $O(128)$ operazioni XOR e shift
2. **Lookup tables:** Pre-calcola multipli di H
3. **Istruzioni SIMD:** CPU moderne hanno istruzioni dedicate (PCLMULQDQ su x86)

Sicurezza di GCM:

- IND-CPA sicurezza dalla modalità CTR
- Autenticazione dalla valutazione polinomiale in $GF(2^{128})$
- **Limite di sicurezza:** Dopo $\sim 2^{32}$ blocchi cifrati con la stessa chiave, la probabilità di collisione nel counter diventa significativa
- **Attacchi nonce-reuse:** Se il nonce è riutilizzato, la sicurezza è completamente compromessa (si può recuperare H e forgiare tag)

ChaCha20-Poly1305:

Combinazione di ChaCha20 (stream cipher) e Poly1305 (MAC).

ChaCha20:

ChaCha20 è uno stream cipher basato sulla costruzione ARX (Add-Rotate-XOR).

Lo stato è una matrice 4×4 di word da 32 bit (totale 512 bit):

```

cccccccc cccccccc cccccccc cccccccc
kkkkkkkk kkkkkkkk kkkkkkkk kkkkkkkk
kkkkkkkk kkkkkkkk kkkkkkkk kkkkkkkk
bbbbbbbb nnnnnnnn nnnnnnnn nnnnnnnn

```

Dove:

- c: costanti ("expand 32-byte k")
- k: chiave a 256 bit (8 word)
- b: block counter (32 bit)
- n: nonce a 96 bit (3 word)

Quarter Round: Operazione fondamentale su 4 word (a,b,c,d):

```

a += b; d ^= a; d <<= 16;
c += d; b ^= c; b <<= 12;

```

$a \oplus= b; d \wedge= a; d \lll= 8;$
 $c \oplus= d; b \wedge= c; b \lll= 7;$

Round completo:

- 4 column rounds (sui 4 quartetti di colonne)
- 4 diagonal rounds (sui 4 quartetti diagonali)

ChaCha20 completo:

- 20 round (10 double-rounds)
- Aggiungi lo stato iniziale allo stato finale (per invertibilità)
- Output 512 bit di keystream

ChaCha20-Poly1305:

1. Usa ChaCha20 per cifrare il plaintext
2. Il primo blocco di keystream genera la chiave per Poly1305
3. Usa Poly1305 per autenticare $AAD \parallel Ciphertext \parallel \text{len}(AAD) \parallel \text{len}(C)$

Vantaggi:

- Velocità eccellente su software
- Nessuna vulnerabilità timing (operazioni a tempo costante)
- Sicurezza dimostrata
- Usato in TLS 1.3, WireGuard, SSH

9. Protocolli Crittografici Avanzati

9.1 Zero-Knowledge Proofs

Un proof system zero-knowledge permette a un prover di convincere un verifier della verità di un'affermazione senza rivelare alcuna informazione oltre alla verità stessa.

Definizione formale:

Un protocollo interattivo (P, V) per una relazione R è zero-knowledge se soddisfa:

1. **Completezza:** Se $(x, w) \in R$, allora V accetta l'interazione con $P(x, w)$ con probabilità vicina a 1
2. **Correttezza (Soundness):** Se $x \notin L$ (dove $L = \{x : \exists w. (x, w) \in R\}$), allora per ogni prover disonesto P^* , V rifiuta con probabilità vicina a 1
3. **Zero-Knowledge:** Esiste un simulatore S che, dato solo x (senza witness w), può produrre una trascrizione dell'interazione indistinguibile da una trascrizione reale

Esempio classico: Protocollo di Schnorr:

Il protocollo di Schnorr per dimostrare la conoscenza del logaritmo discreto.

Setup: Gruppo G di ordine q generato da g . Sia $y = g^x$ dove x è il segreto.

Protocollo:

1. **P** sceglie $r \leftarrow \mathbb{Z}_q$, calcola $t = g^r$, invia t a V
2. **V** sceglie challenge $c \leftarrow \mathbb{Z}_q$, invia c a P
3. **P** calcola $s = r + cx \bmod q$, invia s a V
4. **V** accetta se $g^s = t \cdot y^c$

Completezza:

$$g^s = g^{(r+cx)} = g^r \cdot g^{(cx)} = g^r \cdot (g^x)^c = t \cdot y^c \quad \checkmark$$

Soundness (sketch):

Se il prover può rispondere correttamente a due challenge diversi c e c' , allora:

$$g^s = t \cdot y^c \quad g^{s'} = t \cdot y^{c'}$$

Dividendo:

$$g^{(s-s')} = y^{(c-c')}$$

Quindi:

$$y = g^{((s-s')/(c-c'))}$$

Quindi il prover può estrarre $x = (s-s')/(c-c') \bmod q$.

Zero-Knowledge:

Il simulatore S opera così (simulazione con rewinding):

1. Scegli s, c casualmente
2. Calcola $t = g^s \cdot y^{(-c)}$
3. Output trascrizione (t, c, s)

Questa trascrizione ha la stessa distribuzione di una trascrizione reale! Infatti:

- In una trascrizione reale: $t = g^r, s = r + cx$
- Nella simulazione: $t = g^s \cdot y^{(-c)} = g^s \cdot g^{(-cx)} = g^{(s-cx)} = g^{r'}$ dove $r' = s-cx$

Dato che r (reale) e s (simulato) sono uniformi, le distribuzioni coincidono.

Fiat-Shamir Heuristic:

Trasforma un protocollo interattivo in una firma non-interattiva usando una funzione hash:

Invece di ricevere c dal verifier, il prover calcola:

$$c = H(t \parallel m)$$

dove m è il messaggio da firmare.

Firma risultante: (t, s) dove $s = r + cx \bmod q$

Verifica: Controlla $g^s = t \cdot y^{H(t \parallel m)}$

Sicurezza: Nel modello dell'oracolo casuale, questo è sicuro. In pratica, funziona bene con hash crittografiche.

9.2 Protocolli Multi-Party Computation (MPC)

Il calcolo multi-parte permette a n partecipanti di computare una funzione $f(x_1, \dots, x_n)$ dove il partecipante i conosce solo x_i , senza rivelare gli input agli altri.

Secret Sharing di Shamir:

Schema (t, n) -threshold: un segreto s può essere ricostruito da qualunque insieme di t partecipanti, ma $t-1$ partecipanti non hanno informazione su s .

Costruzione:

1. Scegli un polinomio casuale di grado $t-1$:

$$P(x) = s + a_1x + a_2x^2 + \dots + a_{t-1}x^{t-1}$$

dove s è il segreto e a_i sono coefficienti casuali in $\mathbb{Z}_p$ (p primo)

2. Distribuisci share: il partecipante i riceve $(i, P(i))$

Ricostruzione:

Data un qualunque insieme S di t share $\{(i, y_i) : i \in S\}$, usa l'interpolazione di Lagrange:

$$P(x) = \sum_{(i \in S)} y_i \cdot L_i(x)$$

dove i polinomi di Lagrange sono:

$$L_i(x) = \prod_{(j \in S, j \neq i)} (x - j) / (i - j)$$

Il segreto è:

$$s = P(0) = \sum_{(i \in S)} y_i \cdot L_i(0)$$

Dove:

$$L_i(0) = \prod_{(j \in S, j \neq i)} (0 - j) / (i - j) = \prod_{(j \in S, j \neq i)} (-j) / (i - j)$$

Esempio completo di ricostruzione

Supponiamo (3,5)-threshold con $p = 17$, segreto $s = 10$.

Distribuzione:

Polinomio: $P(x) = 10 + 7x + 3x^2 \pmod{17}$

Share:

- $S_1 = (1, P(1)) = (1, 10 + 7 + 3) = (1, 20 \pmod{17}) = (1, 3)$
- $S_2 = (2, P(2)) = (2, 10 + 14 + 12) = (2, 36 \pmod{17}) = (2, 2)$
- $S_3 = (3, P(3)) = (3, 10 + 21 + 27) = (3, 58 \pmod{17}) = (3, 7)$
- $S_4 = (4, P(4)) = (4, 10 + 28 + 48) = (4, 86 \pmod{17}) = (4, 1)$
- $S_5 = (5, P(5)) = (5, 10 + 35 + 75) = (5, 120 \pmod{17}) = (5, 1)$

Ricostruzione con {1, 2, 3}:

Calcolo dei coefficienti di Lagrange:

$$L_1(0) = (0-2)(0-3)/((1-2)(1-3)) = 6/((-1)(-2)) = 6/2 = 3 \pmod{17}$$

$$L_2(0) = (0-1)(0-3)/((2-1)(2-3)) = 3/(1 \cdot (-1)) = 3/(-1) = -3 \equiv 14 \pmod{17}$$

$$L_3(0) = (0-1)(0-2)/((3-1)(3-2)) = 2/(2 \cdot 1) = 2/2 = 1 \pmod{17}$$

Ricostruzione:

$$\begin{aligned} s &= y_1 \cdot L_1(0) + y_2 \cdot L_2(0) + y_3 \cdot L_3(0) \\ &= 3 \cdot 3 + 2 \cdot 14 + 7 \cdot 1 \\ &= 9 + 28 + 7 \\ &= 44 \equiv 10 \pmod{17} \quad \checkmark \end{aligned}$$

Verifica con {2, 3, 4}:

$$L_2(0) = (0-3)(0-4)/((2-3)(2-4)) = 12/((-1)(-2)) = 12/2 = 6 \pmod{17}$$

$$L_3(0) = (0-2)(0-4)/((3-2)(3-4)) = 8/(1 \cdot (-1)) = -8 \equiv 9 \pmod{17}$$

$$L_4(0) = (0-2)(0-3)/((4-2)(4-3)) = 6/(2 \cdot 1) = 3 \pmod{17}$$

Ricostruzione:

$$\begin{aligned} s &= 2 \cdot 6 + 7 \cdot 9 + 1 \cdot 3 \\ &= 12 + 63 + 3 \\ &= 78 \equiv 10 \pmod{17} \quad \checkmark \end{aligned}$$

Privacy perfetta - Dimostrazione completa

Teorema: Qualunque insieme di $t-1$ o meno share non rivela informazione sul segreto.

Dimostrazione:

Sia S un insieme di $k \leq t-1$ share: $\{(i_1, y_1), \dots, (i_k, y_k)\}$.

Fissiamo un qualunque candidato segreto $s' \in \mathbb{Z}_p$.

Dobbiamo dimostrare: $P(\text{segreto} = s' \mid \text{share osservati}) = P(\text{segreto} = s') = 1/p$

Costruiamo un polinomio $P'(x)$ di grado $t-1$ che:

1. Passa per tutti i k punti osservati: $P'(i_j) = y_j$ per $j = 1, \dots, k$
2. Soddisfa $P'(0) = s'$

Questo è sempre possibile perché abbiamo $k + 1$ condizioni (k punti + valore in 0) e t gradi di libertà (t coefficienti), con $k + 1 \leq t$.

Costruzione esplicita:

Usiamo interpolazione di Lagrange per i k punti, poi aggiungiamo un termine che vale 0 su questi punti ma aggiusta il valore in 0:

$$P'(x) = L(x) + (s' - L(0)) \cdot Q(x)$$

dove:

- $L(x)$ è l'interpolazione di Lagrange per i k punti
- $Q(x) = \prod_{j=1}^k (x - i_j) / \prod_{j=1}^k (0 - i_j)$ è un polinomio che vale 1 in $x=0$ e 0 in $x=i_j$

Poiché $\deg(L) \leq k-1$ e $\deg(Q) = k$, abbiamo $\deg(P') \leq \max(k-1, k) = k \leq t-1$.

Conclusione: Per ogni possibile segreto s' , esiste un polinomio P' compatibile con gli share osservati. Poiché i coefficienti non vincolati (ce ne sono $t-k \geq 1$) erano scelti uniformemente a caso, ogni s' è equiprobabile.

Quindi: **privacy informazionale perfetta**.

Parte VI: Protocolli Multi-Party, Blockchain e Conclusioni

9.2 Protocolli Multi-Party Computation (continuazione)

Interpolazione di Lagrange (continuazione):

$$L_i(x) = \prod_{j \in S, j \neq i} (x - j)/(i - j)$$

Il segreto è $s = P(0) = \sum_{i \in S} y_i \cdot L_i(0)$.

Esempio numerico:

Supponiamo (3,5)-threshold con $p = 17$, segreto $s = 10$.

1. Scegli polinomio casuale di grado 2: $P(x) = 10 + 7x + 3x^2 \pmod{17}$

2. Share distribuiti:

- Partecipante 1: $(1, P(1)) = (1, 20 \pmod{17}) = (1, 3)$
- Partecipante 2: $(2, P(2)) = (2, 36 \pmod{17}) = (2, 2)$
- Partecipante 3: $(3, P(3)) = (3, 58 \pmod{17}) = (3, 7)$
- Partecipante 4: $(4, P(4)) = (4, 86 \pmod{17}) = (4, 1)$
- Partecipante 5: $(5, P(5)) = (5, 120 \pmod{17}) = (5, 1)$

3. Ricostruzione con partecipanti $\{1,2,3\}$:

$$L_1(0) = (0-2)(0-3)/((1-2)(1-3)) = 6/2 = 3 \pmod{17} = 3 \quad L_2(0) = (0-1)(0-3)/((2-1)(2-3)) = 3/(-1) = -3 \pmod{17} = 14 \quad L_3(0) = (0-1)(0-2)/((3-1)(3-2)) = 2/2 = 1 \pmod{17} = 1$$

$$s = 3 \cdot 3 + 2 \cdot 14 + 7 \cdot 1 = 9 + 28 + 7 = 44 \pmod{17} = 10 \quad \checkmark$$

Proprietà matematiche:

1. **Privacy perfetta:** Qualunque insieme di $t-1$ o meno share è informazionalmente indipendente dal segreto. Dimostrazione: dato un insieme di $t-1$ punti e qualunque valore candidato s' , esiste esattamente un polinomio di grado $t-1$ che passa per quei punti e ha $P(0) = s'$.
2. **Omogeneità:** Gli share di $s_1 + s_2$ sono le somme degli share di s_1 e s_2 . Se $P_1(x)$ e $P_2(x)$ sono i polinomi per s_1 e s_2 , allora $P_1(x) + P_2(x)$ è il polinomio per $s_1 + s_2$.
3. **Linearità rispetto alla moltiplicazione per costante:** Gli share di $c \cdot s$ sono c volte gli share di s .

MPC basato su Secret Sharing:

Per computare $f(x_1, \dots, x_n)$:

1. Ogni partecipante i segretamente condivide il proprio input x_i
2. Le parti eseguono un protocollo per valutare f sui valori condivisi
3. Il risultato è rivelato (o mantenuto condiviso)

Esempio: Moltiplicazione sicura:

Per moltiplicare due segreti condivisi a e b :

1. Ogni partecipante i ha share a_i e b_i
2. Ogni partecipante calcola localmente $c_i = a_i \cdot b_i$
3. Ma $\sum c_i \neq a \cdot b$ in generale!

Problema: Il prodotto di due polinomi di grado $t-1$ ha grado $2t-2$, richiedendo $2t-1$ punti per la ricostruzione.

Soluzione (Beaver): Usa triple pre-comutate (u, v, w) dove $w = u \cdot v$:

1. Le parti rivelano $\alpha = a - u$ e $\beta = b - v$
2. Ciascuno calcola localmente: share di $(ab) = \alpha\beta + \alpha v_i + \beta u_i + w_i$

Correttezza:

$$ab = (\alpha + u)(\beta + v) = \alpha\beta + \alpha v + \beta u + uv = \alpha\beta + \alpha v + \beta u + w$$

Poiché α e β sono pubblici e u, v, w sono condivisi, ciascuno può calcolare uno share del risultato.

Complessità delle triple di Beaver:

La generazione di triple (u, v, w) può essere fatta offline prima di conoscere a e b , usando protocolli come oblivious transfer o crittografia omomorfica.

9.3 Crittografia Omomorfica

La crittografia omomorfica permette di computare su dati cifrati senza decifrarli.

Definizione:

Uno schema di cifratura è **omomorfico** rispetto a un'operazione $\oplus$ se:

$$\text{Dec}(\text{Enc}(m_1) \oplus \text{Enc}(m_2)) = m_1 \oplus m_2$$

Livelli di omomorfismo:

1. **Partially Homomorphic (PHE):** Supporta una sola operazione (+ o $\times$) per numero illimitato di volte
2. **Somewhat Homomorphic (SHE):** Supporta entrambe le operazioni per un numero limitato di volte
3. **Fully Homomorphic (FHE):** Supporta entrambe le operazioni per numero illimitato di volte

Esempio: Schema di Paillier (PHE additivo):

Setup:

- Scegli due primi grandi p e q
- $n = pq$, $\lambda = \text{lcm}(p-1, q-1)$
- Scegli $g \in \mathbb{Z}_{n^2}^*$ tale che l'ordine di g sia multiplo di n (tipicamente $g = n + 1$)
- Calcola $\mu = (L(g^\lambda \bmod n^2))^{-1} \bmod n$ dove $L(x) = (x-1)/n$

Chiavi:

- Pubblica: (n, g)
- Privata: (λ, μ)

Cifratura di $m \in \mathbb{Z}_n$:

$$c = g^m \cdot r^n \bmod n^2$$

dove $r \leftarrow \mathbb{Z}_n^*$ è casuale.

Decifratura di c :

$$m = L(c^\lambda \bmod n^2) \cdot \mu \bmod n$$

Proprietà omomorfica:

$$E(m_1) \cdot E(m_2) \bmod n^2 = g^{(m_1) r_1^n} \cdot g^{(m_2) r_2^n} = g^{(m_1+m_2) (r_1 r_2)^n} = E(m_1 + m_2)$$

Quindi la moltiplicazione di crittogrammi corrisponde all'addizione di plaintext!

Correttezza matematica:

Per il teorema di Eulero generalizzato, per ogni $w \in \mathbb{Z}_n^*$:

$$w^{\lambda n} \equiv 1 \pmod{n^2}$$

Si può dimostrare che:

$$g^\lambda \equiv 1 + kn \pmod{n^2} \text{ per qualche } k \text{ coprimo con } n$$

Quindi:

$$c^\lambda = (g^m \cdot r^n)^\lambda = g^{(m\lambda)} \cdot r^{(n\lambda)} \equiv (1 + kn)^m \cdot 1 \pmod{n^2} \equiv 1 + mkn \pmod{n^2}$$

(espansione binomiale troncata)

$$L(c^\lambda) = mkn/n = mk$$

Quindi:

$$m = mk \cdot k^{-1} = mk \cdot \mu \bmod n$$

Applicazioni di Paillier:

1. Voto elettronico: sommare voti cifrati
2. Calcolo di statistiche su dati sensibili
3. Machine learning su dati cifrati (operazioni lineari)

Fully Homomorphic Encryption (Gentry 2009):

Il primo schema FHE si basa su ideali in reticoli.

Idea chiave - Bootstrapping:

1. Costruisci uno schema SHE con decryption circuit "piccolo"
2. Valuta omomorficamente il decryption circuit stesso
3. Questo "rinfresca" il crittogramma, riducendo il noise accumulato
4. Ripeti per numero arbitrario di operazioni

Noise growth:

In schemi basati su LWE, ogni operazione omomorfica aumenta il noise:

- Addizione: noise cresce linearmente
- Moltiplicazione: noise cresce moltiplicativamente

Quando il noise supera una soglia, la decifratura fallisce.

Schema FHE moderno (esempio BGV):

Setup: Anello $R = \mathbb{Z}[x]/(\Phi_m(x), q)$ dove Φ_m è il m-esimo polinomio ciclotomico.

Chiave segreta: Polinomio s con coefficienti piccoli

Chiave pubblica: Coppia (a, b) dove:

- a casuale in R_q
- $b = as + te$ dove e è un errore piccolo

Cifratura di $m \in R_t$ ($t < q$):

1. Codifica m come polinomio
2. $c_1 = au + te_1 + \Delta m$
3. $c_2 = bu + te_2$

dove u, e_1, e_2 sono errori piccoli, $\Delta = Lq/t$.

Decifratura:

Calcola $Lt/q \cdot (c_1 + c_2s)^{-1} \bmod t$

Operazioni omomorfiche:

- **Addizione:** $(c_1, c_2) + (c'_1, c'_2) = (c_1 + c'_1, c_2 + c'_2)$
- **Moltiplicazione:** Più complessa, richiede operazioni di relinearizzazione e modulus switching

Stato dell'arte (2024):

- Librerie pratiche: Microsoft SEAL, HElib, PALISADE
- Prestazioni: ancora 4-6 ordini di grandezza più lente della computazione plaintext
- Applicazioni emergenti: analisi genomica privata, machine learning privato

10. Blockchain e Crittografia

10.1 Funzioni Hash e Proof-of-Work

Proof-of-Work (Bitcoin):

Trova un nonce tale che:

$$H(\text{block_header} || \text{nonce}) < \text{target}$$

dove target determina la difficoltà.

Difficoltà attesa: Se H produce output uniformi di n bit, e $\text{target} = 2^n / D$, allora:

$$E[\text{tentativi}] = D$$

La probabilità di successo per tentativo è:

$$p = \text{target} / 2^n = 1/D$$

Tempo atteso con hash rate H_{rate} :

$$T = D / H_{\text{rate}}$$

Esempio Bitcoin:

- $H = \text{SHA-256}$ ($n = 256$)
- Difficoltà attuale (2024): $D \approx 10^{25}$
- Hash rate rete: $\sim 500 \text{ EH/s} = 5 \times 10^{20} \text{ hash/s}$
- Tempo tra blocchi: $\sim 10 \text{ minuti} = 600 \text{ secondi}$

Verifica: $600 = 10^{25} / (5 \times 10^{20}) \approx 2 \times 10^4 \text{ secondi}$, aggiustato dinamicamente.

Proprietà crittografiche richieste:

1. **Puzzle-friendliness:** Nessun shortcut per trovare soluzioni
2. **Difficoltà verificabile:** Facile verificare che $H(\text{header} || \text{nonce}) < \text{target}$
3. **Progressless:** Ogni tentativo è indipendente (memoryless)

10.2 Firme Digitali nelle Blockchain

ECDSA in Bitcoin:

Curva secp256k1: $y^2 = x^3 + 7$ su $\mathbb{F}_p$ dove $p = 2^{256} - 2^{32} - 977$.

Parametri:

- p : modulo del campo
- n : ordine del punto base G (primo)
- G : punto base

Indirizzo Bitcoin:

1. Genera coppia di chiavi (d , Q) dove $Q = dG$
2. Serializza Q (33 byte compressed)
3. Hash: $h = \text{RIPEMD160}(\text{SHA256}(Q))$
4. Indirizzo = $\text{Base58Check}(\text{version} \parallel h \parallel \text{checksum})$

Firma transazione:

1. Costruisci messaggio m dalla transazione
2. Calcola $z = \text{SHA256}(\text{SHA256}(m))$
3. Genera firma ECDSA (r , s) su z
4. Include firma e chiave pubblica nella transazione

Verifica:

Nodi verificano che la firma corrisponda alla chiave pubblica che corrisponde all'indirizzo.

10.3 Merkle Trees

Un Merkle tree permette verifica efficiente di inclusione di dati in un insieme.

Costruzione:

1. Livello foglie: $H(\text{data}_1)$, $H(\text{data}_2)$, ..., $H(\text{data}_n)$
2. Livello intermedio: $H(H_1 \parallel H_2)$, $H(H_3 \parallel H_4)$, ...
3. Continua finché si raggiunge la radice

Proof di inclusione:

Per dimostrare che data_i è nell'albero:

1. Fornisci data_i
2. Fornisci il "cammino di autenticazione": i sibling hash dal livello foglia alla radice

Dimensione proof: $O(\log n)$ hash

Verifica: $O(\log n)$ computazioni hash

Uso in Bitcoin:

Ogni blocco contiene un Merkle tree delle transazioni. Lo header del blocco include solo la Merkle root. Ciò permette:

- **SPV (Simplified Payment Verification):** Client leggeri possono verificare l'inclusione di una transazione scaricando solo $O(\log n)$ dati invece dell'intero blocco

Matematica della sicurezza:

Per falsificare una proof di inclusione, un attaccante deve:

1. Trovare una collisione in H, oppure
2. Trovare una preimmagine per la radice

Con SHA-256, entrambi sono computazionalmente intrattabili.

11. Teoremi Avanzati e Risultati Recenti

11.1 Il Teorema del Numero Primo e Densità dei Primi

Teorema del Numero Primo:

Sia $\pi(x)$ il numero di primi $\leq x$. Allora:

$$\pi(x) \sim x / \ln(x)$$

Più precisamente:

$$\lim_{x \rightarrow \infty} \pi(x) / (x/\ln(x)) = 1$$

Conseguenze per la crittografia:

La probabilità che un numero casuale di n bit sia primo è approssimativamente:

$$P(\text{n-bit number is prime}) \approx 1 / (n \ln 2) \approx 1.44/n$$

Per $n = 1024$: $P \approx 1/710$

Quindi per generare un primo da 1024 bit:

- Testa random numbers finché uno è primo
- Numero atteso di tentativi: ~ 710
- Con test probabilistico (Miller-Rabin): $\sim 710 \times k$ iterazioni

Distribuzione dei gap tra primi:

Il gap medio tra primi vicini a x è $\sim \ln(x)$.

Per $x \approx 2^{1024}$: gap medio $\approx 1024 \ln(2) \approx 710$

Questo giustifica l'efficienza della generazione di primi: non dobbiamo cercare troppo lontano.

11.2 Complessità Computazionale e Limiti Inferiori

Classi di complessità rilevanti:

- **P**: Problemi risolvibili in tempo polinomiale
- **NP**: Problemi verificabili in tempo polinomiale
- **coNP**: Complementi di problemi in NP
- **BQP**: Problemi risolvibili in tempo polinomiale quantistico

Assunzioni crittografiche:

1. **P $\neq$ NP**: Non dimostrata ma ampiamente creduta
2. **Fattorizzazione $\notin$ P**: Non dimostrata, ma no algoritmo polinomiale noto
3. **DLP $\notin$ P**: Simile alla fattorizzazione
4. **Fattorizzazione $\in$ BQP**: Dimostrato (Shor 1994)

Implicazioni:

Se $P = NP$:

- Tutta la crittografia asimmetrica basata su NP-hard problems collassa
- Anche molti schemi simmetrici diventano insicuri
- La ricerca di collisioni hash diventa efficiente

Limiti inferiori dimostrati:

- Ricerca in spazio non ordinato: $\Omega(\sqrt{n})$ query nel modello quantistico (ottimalità di Grover)
- Ricerca esaustiva: $\Omega(2^n)$ nel modello classico per spazio di n bit

11.3 Teoremi Recenti (2020-2024)**Progressi nella fattorizzazione:**

- Nuove eurystiche per GNFS hanno ridotto costanti, ma complessità asintotica invariata
- Record: 829-bit RSA number fattorizzato (2020)
- Stima: RSA-2048 richiede ancora $\sim 2^{140}$ operazioni

Progressi in curve ellittiche:

- Scoperta di nuove famiglie di curve con proprietà ottimali
- Ristretto255: encoding one-to-one per elementi del gruppo
- Nuovi algoritmi per il pairing computing: $\sim 20\%$ più veloci

Crittografia post-quantistica:

- NIST standardization completata (2024)
- Nuovi attacchi lattice-based: richiedono aumento parametri del 10-15%
- Prove di sicurezza più strette per LWE sotto assunzioni più deboli

Breakthrough in FHE:

- Nuove tecniche di bootstrapping: 10x speedup (2022-2023)
- Schemi ibridi FHE/MPC: migliori tradeoff comunicazione/computazione
- TFHE (Torus FHE): operazioni logiche estremamente veloci

11.4 Congetture Aperte**Congettura di Goldbach:**

Ogni numero pari > 2 è somma di due primi.

Rilevanza crittografica: Se falsa, potrebbe esistere una struttura nascosta nei primi sfruttabile per attacchi.

Congettura dei primi gemelli:

Esistono infiniti primi p tali che $p+2$ è anch'esso primo.

Rilevanza: Distribuzione fine dei primi influenza sicurezza di alcuni schemi.

Ipotesi di Riemann:

$\zeta(s) = \sum_{n=1}^{\infty} 1/n^s$ ha tutti i suoi zeri non banali sulla retta $\text{Re}(s) = 1/2$.

Rilevanza: Fornisce bounds migliori su $\pi(x)$ e sulla distribuzione dei primi, influenzando analisi di sicurezza probabilistica.

Congettura sulla difficoltà del DLP:

Non esiste algoritmo classico polinomiale per il DLP in gruppi generici.

Status: Supportata da risultati di lower bound nel modello del gruppo generico, ma non dimostrata per gruppi specifici come $(\mathbb{Z}/p\mathbb{Z})^*$.

12. Conclusioni e Prospettive Future

12.1 Sintesi delle Basi Matematiche

La crittografia moderna si fonda su diversi pilastri matematici interconnessi:

1. **Teoria dei Numeri:** Fornisce i problemi difficili (fattorizzazione, DLP) e gli strumenti algebrici (teoremi di Fermat ed Eulero, TCR)
2. **Algebra Astratta:** Strutture di gruppo, anelli e campi forniscono il framework per curve ellittiche, costruzioni hash, e schemi avanzati
3. **Teoria della Complessità:** Distingue tra problemi trattabili e intrattabili, giustificando le assunzioni di sicurezza
4. **Teoria dei Reticoli:** Base per la crittografia post-quantistica, con proprietà di worst-case to average-case
5. **Teoria dell'Informazione:** Quantifica sicurezza informazionale, fornisce limiti fondamentali (perfect secrecy di Shannon)

La matematica della crittografia moderna rappresenta una delle sintesi più eleganti e potenti tra teoria pura e applicazione pratica. Teoremi dimostrati migliaia di anni fa (come quello di Fermat) sono oggi fondamentali per la sicurezza delle comunicazioni globali. Allo stesso

tempo, problemi aperti da secoli (come la distribuzione dei primi) continuano a influenzare il design di sistemi crittografici.

La bellezza della crittografia risiede nella sua natura dual-purpose: da un lato fornisce strumenti pratici essenziali per la società digitale; dall'altro stimola ricerca matematica profonda in aree che vanno dalla teoria dei numeri alla geometria algebrica, dalla teoria della complessità alla fisica quantistica.

L'avvento dei computer quantistici rappresenta sia una minaccia che un'opportunità: mentre costringe a rivedere decenni di infrastruttura crittografica, spinge anche lo sviluppo di nuove matematiche (reticoli, codici) e di nuove tecniche (crittografia omomorfa, MPC) che promettono applicazioni ben oltre la semplice sostituzione degli schemi attuali.

Il futuro della crittografia sarà probabilmente caratterizzato da maggiore diversità: non un singolo paradigma dominante, ma un ecosistema di tecniche complementari, ciascuna ottimizzata per specifici scenari d'uso, ciascuna basata su diversi pilastri matematici, ciascuna con diversi profili di sicurezza e prestazioni.

La matematica rimarrà sempre il fondamento insostituibile: ogni bit di sicurezza in un sistema crittografico è ultimamente giustificato da un teorema, ogni implementazione sicura deve rispettare vincoli matematici profondi, ogni nuovo schema deve essere analizzato con rigore per garantire che non ceda sotto gli inevitabili attacchi futuri.

In definitiva, la crittografia moderna è la testimonianza del potere della matematica pura di plasmare il mondo reale: equazioni astratte governano la sicurezza delle transazioni finanziarie, teoremi sui gruppi proteggono le comunicazioni private, e la struttura dei reticoli potrebbe essere la chiave per preservare la privacy nell'era quantistica.